Principe de transduction sémantique pour l'application de théories d'interfaces sur des documents de spécification

Aurélien Lamercerie

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES 1
COMUE UNIVERSITÉ BRETAGNE LOIRE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Par

« Aurélien LAMERCERIE »

« Principe de transduction sémantique pour l'application de théories d'interface sur des documents de spécification »

Thèse présentée et soutenue à « Rennes », le « jeudi 8 avril 2021 »
Unité de recherche : IRISA, Université de Rennes 1

Rapporteurs avant soutenance :

Iulian OBER	Maitre de conférences, IRIT, Université de Toulouse
Christian RETORE	Professeur, LIRMM, Université de Montpellier

Composition du Jury :

Présidente :	Sophie PINCHINAT	Professeur, IRISA, Université de Rennes 1
Examinateurs :	Iulian OBER	Maitre de conférences, IRIT, Université de Toulouse
	Christian RETORE	Professeur, LIRMM, Université de Montpellier
	Emmanuel LEDINOT	Ingénieur, THALES
Dir. de thèse :	Benoit CAILLAUD	Directeur de recherche, INRIA
Co-dir. de thèse :	Annie FORET	Maitre de conférences, IRISA, Université de Rennes 1

Remerciements

Ce manuscrit marque la dernière étape d'une belle aventure scientifique, dont la réalisation n'aurait pas été possible sans appui.

En premier lieu, je tiens à remercier mes directeurs de thèse, Annie Foret et Benoit Caillaud, pour la confiance qu'ils m'ont accordé et pour l'encadrement qu'ils m'ont offert. J'ai apprécié leurs conseils au cours de nos multiples réunions de travail.

J'adresse également toute ma reconnaissance à mon jury de thèse, dont les remarques pertinentes m'ont permis d'améliorer ce document. Je remercie ainsi Iulian Ober et Christian Rétoré, pour l'évaluation de ce manuscrit dans une période difficile. Je remercie aussi Emmanuel Ledinot et Sophie Pinchinat, qui ont accepté de participer à mon jury, en y apportant un parfait complément. De même, je tiens à remercier Denis Béchet, Jean-Baptiste Raclet et Olivier Ridoux, pour leur accompagnement dans le cadre du comité de suivi, et Christian Boitet, pour les commentaires apportés à ce document.

Je souhaiterais remercier, en sus, mes collègues des équipes SemLis et Hycomes, ainsi que les membres du projet S3PM/SunSet. Je garde le souvenir de moments d'échanges aussi enrichissants que plaisants. J'ai eu l'occasion, en complément de mon activité de recherche, d'enseigner à l'université et de participer à l'encadrement de stages. Je suis reconnaissant de la confiance qui m'a été accordée à ce titre. En outre, je remercie Matthieu Robert, dont le stage a contribué positivement à ma réflexion tout en complétant mes données expérimentales.

Au delà, j'exprime toute ma reconnaissance aux personnes, trop nombreuses pour être citées, qui m'ont aidé à un moment ou un autre de cette thèse. Cette expérience a donné lieu à de multiples rencontres, fructueuses et instructives.

Finalement, je remercie ma famille, Amandine et mes trois enfants, dont le soutien aura apporté une agréable légèreté à cette délicate entreprise.

TABLE DES MATIÈRES

Chapitre 1

Introduction

Le développement de systèmes capables de comprendre les langues naturelles est un défi majeur. Il implique la prise en compte de différents aspects renvoyant à une somme d'aptitudes anthropiques. Il en est ainsi de notre capacité à distinguer des entités ou des événements dans un texte, et à en comprendre les relations. Cette tâche est complexe à traduire en termes formels et intégrables dans un processus automatisé. Un long chemin reste à parcourir, malgré les avancées récentes apportées par les méthodes d'apprentissage profond. Notre ambition est de faire quelques pas sur cette voie, et de contribuer au franchissement d'une nouvelle étape.

L'enjeu de ce mémoire est ainsi posé. La cible est un processus d'analyse pour une application sur des documents textuels, rédigés dans une langue naturelle telle que l'anglais. La mise en œuvre visée est une chaîne de traitements, automatisée de bout en bout, et intégrant des facultés d'interprétation et de raisonnement sur les données traitées. Précisément, nous proposons d'étudier la façon de relier des énoncés de langue anglaise à des modèles formels exploitables dans un cadre théorique adapté.

Le contexte applicatif porte sur les systèmes cyberphysiques, c'est-à-dire tout type de système dans lequel des composants logiciels interagissent étroitement avec un environnement physique. La conception de tels systèmes s'appuie sur des cahiers des charges. Ces documents consistent en un ensemble structuré d'énoncés, appelés exigences. Leur fonction est de définir les contraintes et les caractéristiques attendues des composants et de leur environnement. Il s'agit d'une ressource fondamentale pour les acteurs d'un projet. Leur rédaction intervenant très tôt dans le cycle de développement, le coût d'une erreur non détectée à ce stade est potentiellement important. Ces documents peuvent également avoir une valeur juridique. Leur validation en est d'autant plus essentielle.

Plusieurs méthodes formelles ont été proposées pour aider les industriels dans cette tâche, telles que les logiques temporelles [1, 2] ou les automates d'interface [3]. Rigoureusement fondées, elles promettent un support potentiellement performant, du moins en théorie. En pratique, elles n'ont pas été largement adoptées, pour des raisons essentiel-

lement pragmatiques telles que le manque d'expertise appropriée ou d'outil adéquat. De fait, l'usage de ces techniques implique de traduire les spécifications dans des formalismes spécifiques dont la prise en main n'a rien de trivial.

La logique de notre démarche est d'aboutir à la proposition de logiciels favorisant une bonne acceptation des utilisateurs. Un cahier des charges peut se ramener à un ensemble de propriétés se rapportant aux composants d'une architecture. Dès lors, est-il possible de vérifier la cohérence et la complétude de cet ensemble ?

La problématique est double. D'un côté, nous voulons extraire les informations utiles pour formaliser des énoncés exprimés dans une langue naturelle. De l'autre côté, il s'agirait d'exploiter les formalisations obtenues en proposant un modèle de spécification formelle s'inscrivant dans un cadre de raisonnement performant.

Dans le cas des exigences d'un cahier des charges, le but est d'apporter une aide au concepteur dès les prémisses du cycle de développement. L'objectif serait de pouvoir vérifier la cohérence et l'exhaustivité des exigences, pour toute spécification, en gardant une complexité algorithmique raisonnable. Différentes approches sont envisageables. Elles ont généralement en commun de soutenir la conception avec une démarche compositionnelle, incluant les notions d'abstraction, de raffinement et de composition. La conception basée sur les contrats [4] offre un cadre formel unifié pour de telles méthodes en tant que théorie générique. Nous nous inscrivons dans ce cadre pour développer une théorie autour d'un formalisme offrant un bon compromis en termes d'expressivité pratique, de propriétés algébriques et de complexité algorithmique.

Contributions

Nos travaux apportent plusieurs contributions en réponse aux problématiques soulevées. Premièrement, nous dessinons les contours d'un cadre d'analyse par transduction sémantique [5, 6]. Les principes avancés tendent vers l'extraction et la formalisation d'énoncés du langage naturel, qu'il s'agit ensuite d'exploiter. Conséquemment, nous proposons des modèles de spécification adaptés au raisonnement par contrat [7, 8]. Cette méthodologie peut s'appliquer pour accompagner la conception de systèmes. Elle s'exprime en reliant formellement des hypothèses sur les environnements envisageables et des garanties sur les implémentations possibles. La définition d'une théorie de contrats se caractérise ainsi par un ensemble de concepts, d'opérations et de propriétés dont la mise en œuvre permet d'assister la production de cahiers des charges.

L'usage de structures sémantiques autorise la construction de représentations pivot, que nous proposons d'exploiter. Partant de celles-ci, nous définissons un procédé d'analyse que nous nommons principe de transduction sémantique. L'idée centrale se traduit par une série de transformations sur l'interprétation d'un graphe sémantique, dont l'exécution est guidée à l'aide de schémas de transduction. Cette technique, qui s'applique à toute structure pouvant se ramener à un graphe étiqueté, ouvre ainsi la voie à la composition de processus simples, lisibles et adaptables pour l'interprétation d'énoncés de langue naturelle.

Les automates moniteurs [9, 10] définissent des systèmes de transitions d'états où chaque état est associé à une formule propositionnelle, et où certains états sont signalés comme illégaux. Ils autorisent la modélisation de spécifications prenant en compte une certaine variabilité de réalisations possibles. Une théorie de contrats peut être construite autour de ce formalisme, avec une algèbre de composition assez classique, mais dont les opérateurs nécessitent néanmoins quelques ajustements.

Les automates à ensembles d'acceptation propositionnels, que nous introduisons au chapitre 8, constituent un formalisme alternatif. Celui-ci peut être vu comme une extension des moniteurs où les états illégaux seraient remplacés par des ensembles spéciaux, appelés ensembles d'acceptation. Cette nouvelle structure combine des propriétés d'états et des propriétés temporelles simples. Une algèbre de composition peut lui être associée. Une théorie complète est ainsi produite, avec une relation de raffinement et différents opérateurs utiles (composition parallèle, conjonction et quotient). De fait, la théorie obtenue supporte des méthodes de raisonnement performantes pour l'analyse d'exigences comportementales.

Grille de lecture

Ce mémoire se compose de dix chapitres dont la lecture peut suivre trois axes complémentaires. Il est ainsi possible de dégager trois blocs. Le premier bloc apporte un éclairage d'ensemble à la problématique traitée. Il inclut le chapitre introductif, les chapitres 2 et 9 et la conclusion. Le second bloc considère des questions d'analyse du langage naturel à travers deux chapitres, les chapitres 3 et 4. Finalement, le dernier bloc s'étend sur quatre chapitres, du chapitre 5 au chapitre 8, pour traiter de la mise en œuvre de raisonnement par contrat avec des modèles adaptés.

Cette introduction a permis de poser la problématique traitée en des termes génériques.

Le contexte est précisé dans le chapitre 2. En réponse, la proposition d'un processus de bout en bout est défendue dans les chapitres 2 et 9. Le premier argumente en faveur de briques intégrables dans la chaîne de traitement. Le second déroule le processus exposé sur deux exemples complets. Un développement détaillé des techniques envisagées est proposé dans les autres chapitres.

Le premier axe avancé est un processus automatique qui utilise le traitement du langage naturel pour extraire et formaliser des informations utiles de documents de spécification. Notre proposition peut être observée suivant deux directions. La première, tournée vers la source, est présentée au chapitre 3. Elle vise la construction de représentations sémantiques formelles capturant le sens des énoncés. La seconde, dirigée vers la cible, est définie dans le chapitre 4. Elle se base sur le principe de transduction sémantique, une technique d'interprétation et de transformation applicable à des graphes sémantiques.

Le deuxième domaine d'étude de cette thèse s'inscrit dans le champ de l'ingénierie des exigences. Le chapitre 5 pose les bases de construction d'une théorie d'interface. Une telle théorie constitue un cadre adapté à l'analyse d'exigences comportementales exprimées à un stade précoce du processus de conception. Cette approche associe un modèle de réalisation, un modèle de spécification et une algèbre de composition. Préalablement à nos propositions, les structures de Kripke sont ainsi distinguées comme modèle de réalisation permettant de décrire les comportements attendus d'un système donné. Les chapitres suivants correspondent plus spécifiquement à notre contribution dans ce domaine. Les ensembles d'acceptation, caractérisés au chapitre 6, permettent d'exprimer la variabilité de réalisations induite par une règle comportementale. Finalement, les chapitres 7 et 8 décrivent deux modèles de spécification, les moniteurs et les DPAA. Ces formalismes sont associés au modèle de Kripke pour établir une théorie d'interface complète.

Chapitre 2

Assistance à la conception de systèmes

Notre sujet d'étude porte sur les systèmes techniques au sens large, où les éléments informatiques concourent au contrôle d'entités physiques. L'architecture de tels systèmes intègre des composants logiciels et physiques qui fonctionnent en étroite interaction. On les retrouve dans des domaines très variés, tels que l'aéronautique, le ferroviaire, l'énergie ou encore l'automobile. L'efficacité et la fiabilité de ces systèmes, dénommés systèmes cyber-physiques (CPS [1]), n'ont cessé de progresser, suivant le perfectionnement des méthodes de conception et des connaissances scientifiques.

La conception de tels systèmes, reposant sur plusieurs composants en interaction, est naturellement complexe. Le développement peut ainsi être délégué à des équipes indépendantes. D'un point de vue méthodologique, il apparaît nécessaire de spécifier rigoureusement les caractéristiques attendues. En pratique, le cahier des charges regroupent les propriétés attendues sur le fonctionnement du système et les hypothèses faites sur l'environnement dans lesquels les propriétés doivent être satisfaites. Ces documents associent des ensembles de prescriptions à vérifier, nommées exigences.

Les premières étapes de conception permettent d'aboutir à la rédaction de ces documents. Elles sont donc critiques. De fait, détecter des erreurs de spécification aussi tôt que possible permet d'économiser beaucoup de temps, d'efforts et d'argent. Malheureusement, l'état de l'art actuel ne permet pas d'analyser formellement les exigences par des méthodes automatiques, ou même assistées par ordinateur. Des méthodes existent, basées par exemple sur les logiques temporelles ou des formalismes à base d'automates, mais elles ne sont pas applicables sur les énoncés en langue naturelle. Elles nécessitent une étape de formalisation complexe, et n'ont pas été, de ce fait, largement adoptées.

Notre contribution vise à dépasser cet état de fait. Nous proposons les bases d'un processus automatique combinant des techniques de traitement du langage naturel et des

1. CPS : Cyber-Physical System.

formalismes de haut niveau. Partant de cahiers des charges, nous visons l'extraction et la formalisation automatique des exigences, tout en apportant certaines garanties pour contrôler les résultats obtenus. La finalité est d'être en mesure de mettre en œuvre des outils visant à garantir la consistance et la complétude de documents de spécification. Plus précisément, l'enjeu serait d'aider le concepteur à s'assurer de l'absence de contradiction entre les exigences, avec néanmoins une couverture de tous les comportements attendus.

Le reste de ce chapitre s'organise comme suit. Le contexte est précisé dans la section 2.1. Différents travaux connexes, exploitables pour la formalisation d'exigences, sont présentés section 2.2. Finalement, la section 2.3 expose une réponse générale à la problématique posée, sous la forme d'une chaîne de traitement de bout en bout.

2.1 Contexte

L'analyse des besoins est la première étape du cycle de développement d'un système. Il s'agit de définir précisément le contexte applicatif et les propriétés fonctionnelles que le système devra vérifier. Les cahiers des charges forment l'élément central de cette étape. Leurs fonctions sont de rassembler les contraintes et les caractéristiques attendues. Ces documents consistent en une série d'énoncés, appelés exigences, classiquement exprimés en langage naturel.

Cette ressource est fondamentale pour les acteurs d'un projet. Leur validation est essentielle, car les conséquences d'une erreur non détectée peuvent s'avérer coûteuses. Des erreurs à ce niveau conduisent à des retards de livraison importants, et des imperfections en production potentiellement critiques. Les cahiers des charges se déclinent également sous la forme de documents contractuels. Il en résulte des actes avec une valeur juridique, qui se doivent d'être cohérents et complets.

Au delà, la conception de systèmes cyber-physiques résulte de l'assemblage de différents composants. Ces éléments sont généralement conçus par des équipes multidisciplinaires indépendantes. Chaque équipe doit non seulement spécifier le comportement attendu du composant en cours de conception, mais également énoncer les hypothèses sous lesquelles le composant doit fonctionner. Au moment de l'intégration, ces hypothèses doivent être vérifiées par rapport au comportement du système pris dans sa globalité.

Ainsi, un cahier des charges peut se ramener à une liste d'énoncés, appelés exigences. Chacun de ces énoncés se compose d'une ou plusieurs phrases pour définir une contrainte, ou une nécessité, que le système doit satisfaire. Plusieurs types d'exigences peuvent être

révélés, dont les exigences de sûreté, qui portent sur les comportements et les enchaînements d'événements en fonction de l'état du système. Nos travaux se sont focalisés en particulier sur cette catégorie d'exigences, notamment reliées aux modalités exprimant un caractère d'obligation ou d'interdiction (modalités déontiques). Par exemple, le cahier des charges d'un système d'accès pour un parking de voitures pourrait contenir des règles telles que :

— (E_1) When a ticket has been issued, the entry gate must open.

— (E_2) The passage of a car is prohibited if the security gate is closed.

Dans l'exigence E_1, un évènement est obligatoirement attendu (l'ouverture d'une barrière) lorsqu'une certaine propriété est vérifiée (l'émission d'un ticket). A contrario, l'exigence E_2 spécifie une interdiction, celle du passage d'un véhicule, si le système vérifie une autre propriété (la barrière de sécurité est fermée). Pour des systèmes logiciels complexes, la description de ces directives comportementales est une opération délicate. Elle n'en est pas moins importante et indispensable dans une démarche de conception rationnelle et efficace [11].

Ainsi, de nombreuses méthodes ont été développées pour aider les industriels dans cette tâche. L'enjeu est la construction d'un ensemble d'exigences complet et consistant, ne comportant pas d'énoncés incorrects, contradictoires ou redondants. En réponse, différentes techniques d'analyse formelle ont émergé, telle que la méthode SCR [2] [12]. Les logiques temporelles, LTL et CTL [1, 2], peuvent également être exploitées dans ce but. Les automates d'interfaces [3] et les interfaces modales [13, 14] visent à fournir une spécification fusionnée du comportement attendu et des environnements possibles. Les théories de contrats [4], avec une prise en charge native dans plusieurs langages de programmation [15], offrent un cadre flexible en tant que théorie générique.

Le bénéfice potentiel de ces méthodes est perceptible. Pourtant, elles n'ont pas été largement adoptées. Cela est dû en partie à un certain nombre d'obstacles pratiques, tels que la nécessité d'une expertise appropriée, le besoin en formation spécifique ou l'absence d'outillage support adapté. Il y a un écart important entre le bénéfice de ces méthodes en théorie, et leur intégration en pratique. Nos travaux visent à réduire cet écart par l'application de méthodes automatiques appropriées, partant des documents de spécification pour en garantir la consistance et la complétude.

La consistance d'une spécification traduit l'absence d'incohérence. Elle peut s'obtenir, par construction, en combinant les exigences à l'aide de structures aux propriétés adé-

2. SCR : Software Cost Reduction

quates. Les spécifications modales [13], sur lesquelles nous reviendrons plus loin, constituent un exemple de structures pertinentes dans ce but. La bibliothèque Mica [16], basée sur ce formalisme, offre ainsi la possibilité de vérifier formellement des propriétés pour des systèmes réactifs, dont la cohérence des spécifications.

La complétude ne peut pas être formellement vérifiée, mais il est possible, à partir d'une spécification formelle, de simuler différentes réalisations d'un système. L'idée serait de permettre à l'ingénieur-concepteur d'interagir avec sa spécification, avec un système mettant en évidence les comportements possibles ou interdits, et les propriétés des états atteignables à la suite de différentes trajectoires (enchaînements d'événements).

Pour favoriser une bonne acceptation des utilisateurs, la proposition de logiciels intégrables aux outils existants devrait être privilégiée. Cet objectif tend à exclure l'utilisation d'un langage de domaine spécifique (DSL). Un cahier des charges peut se ramener à un ensemble de propriétés se rapportant aux composants d'une architecture. Partant de cet ensemble, l'objectif serait d'aboutir à une représentation formelle des comportements attendus.

Sur le plan opératoire, ce problème peut se décomposer en deux grandes phases. La première est l'analyse formelle des exigences, qui doit permettre d'aboutir à leur représentation sémantique. La seconde consiste en la composition de ces exigences pour obtenir une représentation des comportements possibles du système spécifié. En définitive, l'enjeu est d'apporter une aide aux concepteurs de systèmes techniques, en permettant de vérifier, par des méthodes automatiques ou assistées, des cahiers des charges exprimés en langue naturelle. La réponse envisagée est une chaîne de traitement de bout en bout combinant différentes briques logicielles, répondant aux contraintes posées par les deux phases opératoires. La construction de ces briques soulève plusieurs obstacles, que nous tentons de dépasser dans la suite. La première barrière levée est celle du traitement d'énoncés du langage naturel, que nous proposons d'analyser dans une démarche contrôlée, telle que développée dans les chapitres 3 et 4. Le second défi est celui de la définition d'une théorie formelle adaptée pour une application efficace à différents niveaux d'abstraction.

2.2 Formalisation des exigences

Le passage de spécifications en langue naturelle vers des spécifications formelles suscite un intérêt scientifique évident, à même d'apporter un concours précieux aux développements industriels. Plusieurs lignes directrices peuvent être dégagées, que nous proposons

d'explorer en suivant trois orientations possibles.

Une interface peut être proposée pour faciliter la construction de modèles vérifiables. Cette approche est présentée dans la section 2.2.1. Les *patterns* de spécification suivent cette logique en guidant le concepteur avec des schémas structurés. Ils facilitent ainsi l'élaboration de modèles formels, dont les propriétés peuvent être vérifiées, sans permettre néanmoins l'analyse directe des documents de spécification.

La section 2.2.2 propose un procédé autorisant la construction de représentations logiques en suivant la décomposition d'un énoncé. Les grammaires catégorielles combinatoires (CCG [3], [17]) peuvent servir dans ce but. Elles offrent la possibilité de décomposer une phrase en catégories mettant en évidence les liens syntaxiques du langage traité. Des règles de dérivation sont définies pour combiner les éléments d'une phrase. Reliées à des représentations sémantiques, celles-ci permettent de calculer la représentation formelle d'exigences exprimées en langage naturel.

Le principe d'analyse reposant sur une représentation pivot est généralisé dans la section 2.2.3, avec la présentation de quelques structures adaptées. Cette voie est prolongée dans nos travaux, comme nous le verrons dans la section 2.3 et les prochains chapitres, en tirant notamment parti de représentations sémantiques sous la forme de graphes étiquetés.

2.2.1 Approches basées sur le formalisme cible

Une première orientation consiste à s'appuyer sur le formalisme cible, par exemple la logique temporelle. Le but recherché, en suivant cet axe, est d'ajouter une interface pour faciliter l'expression de propriétés à vérifier à l'aide d'une représentation se rapprochant d'un langage naturel.

Les *patterns* de spécification [18, 19] ont été conçus dans cette optique. Dotés de correspondances avec différents formalismes, ils définissent un ensemble d'abstractions paramétrables, plus facile à appréhender car basé sur le langage naturel. Les schémas proposés autorisent ainsi la spécification de propriétés critiques, telles que la restriction d'occurrences de certains événements ou l'interdiction d'une propriété dans des situations données. De ce fait, ils apportent un support pour un ensemble de techniques permettant de prouver des propriétés sur des modélisations de systèmes.

Les *patterns* de spécification peuvent être associés à différentes représentations, par exemple les systèmes de transitions à états finis, les systèmes temporisés ou les systèmes

3. CCG : Combinatory Categorial Grammar.

hybrides. Des correspondances sont notamment apportées pour les logiques temporelles LTL [4] [20], CTL [5] [2] et les expressions régulières quantifiées QRE [6] [21].

De fait, un système de *patterns* est un ensemble de schémas. Ceux-ci peuvent être regroupés en fonction du formalisme de spécification, par exemple LTL ou CTL. Il est également possible d'avoir une hiérarchie selon le type de comportement décrit, comme montré par la figure 2.1. Dans cette classification, les modèles d'occurrence (Occurrence Patterns) caractérisent un événement pendant l'exécution du système, tandis que les modèles d'ordonnancement (Order Patterns) concernent l'ordre relatif dans lequel plusieurs événements se produisent.

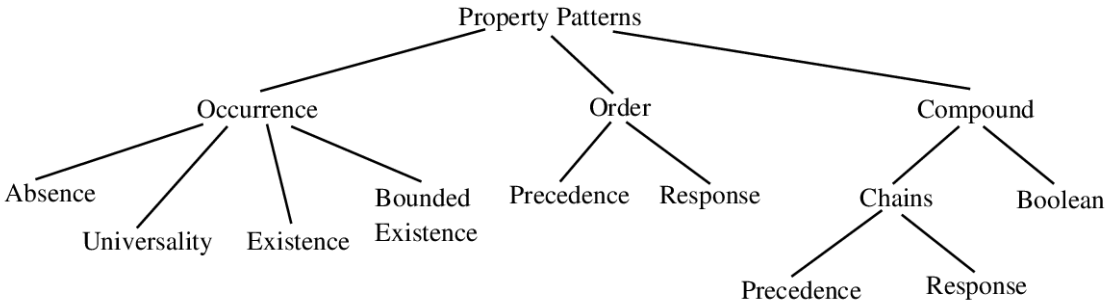


FIGURE 2.1 – Hiérarchie des *patterns* en fonction du type de comportement décrit [18]

Précisément, un *pattern* est une description structurée d'un comportement de système. Il se caractérise par un nom, un énoncé précis de son but (le type de comportement décrit) et une portée (*scope*). Il est également associé à des transcriptions dans quelques formalismes de spécification et à des exemples d'utilisation. Par exemple, le pattern *Response* autorise la description d'une relation de cause à effet entre une paire d'événements, avec une certaine portée. La portée permet de situer le comportement décrit globalement, avant ou après un autre évènement ou en fonction de l'occurrence de deux autres événements.

Une approche analogue est proposée pour la spécification de systèmes en temps réel [22]. Associée à cette méthode, une grammaire anglaise structurée a été développée pour faciliter l'usage des patterns de spécification. Elle prend en charge des modèles avec plusieurs champs d'application, qualitatif (par exemple l'absence, la globalité ou l'enchaînement d'événements) et quantitatif (incluant des informations sur le temps). Les modèles de

4. LTL : Linear Temporal Logic
5. CTL : Computational Tree Logic
6. QRE : Quantified Regular Expression

spécification proposés peuvent être reliés à plusieurs logiques temporelles en temps réel, telles que la logique temporelle métrique MTL [7] [23] ou la logique d'arbres de calcul temporisé TCTL [8] [24].

L'usage d'un langage naturel contrôlé, tel que le langage ACE [9] [25], peut également être envisagé. La structure des phrases y est contrainte avec des règles de construction et d'interprétation, tout en restant très proche de l'anglais courant. Associé à un moteur d'analyse, l'APE [10], il permet de générer des représentations sémantiques sous la forme de structures de représentation du discours (DRS [11]) [26] et d'expressions logiques du premier ordre. Par extension, les DRS peuvent ensuite être traduites dans diverses représentations telles que OWL [12] [27], SWRL [13] [28] ou RuleML [29]. Le contrôle du langage par des processus automatiques est renforcé, au prix d'une perte d'expressivité et d'un temps d'apprentissage. Si les méthodologies construites autour de ces langages ne permettent pas d'analyser directement des énoncés exprimés en langue naturelle, elles offrent néanmoins la possibilité de construire efficacement des éléments d'interfaces entre les documents de spécification et leurs modélisations.

2.2.2 Approches basées sur une décomposition des énoncés

Une seconde orientation peut être suivie en se basant sur une décomposition des énoncés textuels. Cette stratégie vise la construction de structures formelles aux propriétés vérifiables. Elle implique le choix d'une représentation cible avec les caractéristiques adéquates, c'est à dire le bon niveau d'expression et les bonnes propriétés algébriques. Nous verrons, dans le chapitre 5, les bases de construction d'une théorie couvrant cet objectif. Le principal défi reste néanmoins la définition d'une méthode automatique complète pour faire émerger une spécification formelle à partir d'énoncés informels, dépassant de ce fait l'état de l'art actuel.

Suivant les travaux de Montague [30], le sens d'une phrase peut être considéré comme lié à sa construction syntaxique. Le comportement syntaxique de l'énoncé cible peut ainsi être défini en partant d'une décomposition de l'énoncé à l'aide d'un formalisme gramma-

7. MTL : Metric Temporal Logic
8. TCTL : Timed Computational Tree Logic
9. ACE : Attempto Controlled English
10. APE : Attempto Parsing Engine
11. DRS : Discourse Representation Structure
12. OWL : Ontology Web Language
13. SWRL : Semantic Web Rule Language

tical. L'idée est ensuite de relier cette décomposition à une représentation formelle.

Les grammaires catégorielles peuvent servir dans ce but. Ce formalisme de haut niveau pour l'analyse de textes a été introduit au début des années 1950, à partir des travaux de Adjukiewicz [31], Bar-Hillel [32] et Lambek [33]. Il repose sur la notion de catégories primitives et dérivées (on parle aussi de types), offrant la possibilité de décomposer une phrase en catégories. Les catégories composées mettent en évidence les liens syntaxiques du langage ciblé. Les grammaires catégorielles intègrent également des règles de dérivation qui permettent de combiner les éléments d'une phrase.

L'expressivité des grammaires catégorielles classiques est équivalente à l'expressivité des grammaires non-contextuelles [32, 34]. Plusieurs extensions ont été proposées pour dépasser cette contrainte, dont les grammaires catégorielles combinatoires (CCG), introduite par Mark Steedman en 1989 [17]. Ce formalisme utilise un ensemble d'opérateurs pour saisir d'autres aspects linguistiques. Dans cette grammaire, la sémantique des mots est guidée tout au long de l'arbre de dérivation en combinant les sémantiques des autres mots jusqu'à produire la sémantique complète de la phrase, à la Montague [30]. Elle est particulièrement intéressante pour ses liens avec le lambda-calcul, qui pourrait être une représentation formelle intermédiaire avant d'aboutir à d'autres représentations logiques.

La figure 2.2 montre un exemple d'analyse avec le parseur EasyCCG [35] pour la phrase "After a ticket is inserted, the gate may open". L'analyse syntaxique doit permettre de réduire la séquence des catégories associées aux mots à la catégorie principale de la phrase S, ce que nous observons sur cette exemple. Les éléments entre crochets correspondent à des étiquettes grammaticales permettant d'affiner l'analyse syntaxique.

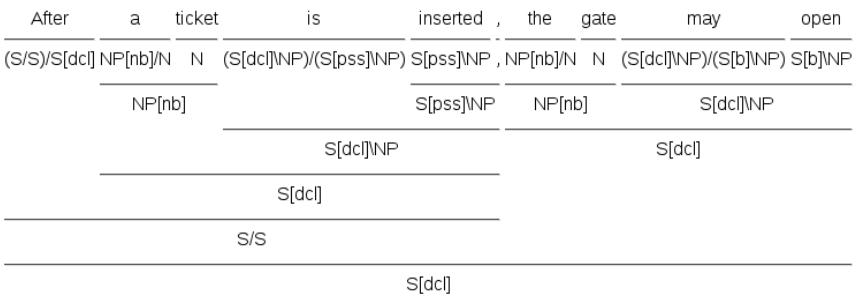


FIGURE 2.2 – Dérivation CCG de la phrase "After a ticket is inserted, the gate may open" construite avec le parser EasyCCG [35]

Les dérivations obtenues avec une analyse CCG peuvent être reliées à une représentation sémantique, en s'appuyant sur les associations définies dans le lexique. Des algo-

rithmes d'analyse CCG en temps polynomial existent [36]. La mise en œuvre de ce modèle s'est traduite dans le développement de plusieurs outils, dont l'outil Boxer/C&C [37]. Un système de résolution de problèmes mathématiques, acceptant la saisie en langage naturel, a également été conçu en se basant sur les CCG [38]. Les Grammaires Catégorielles Abstraites[39] constituent un autre formalisme qui généralise les grammaires catégorielles en traitant directement les catégories et le lambda-calcul.

2.2.3 Approches basées sur une structure pivot

Le passage par une représentation pivot est une troisième orientation possible. Cette idée est présente, par exemple, dans des travaux visant la génération de requêtes SQL ou Sparql à partir de langues naturelles [40]. Elle a été reprise par plusieurs projets, s'appuyant sur des structures variées.

Dans les travaux de *Sadoun et al.*, une modélisation des connaissances, basée sur une ontologie OWL-DL et une logique de description, est ainsi exploitée comme modèle pivot [41]. Celui-ci permet de relier des énoncés à des spécifications formelles en identifiant des instances de propriétés.

Proposée au début des années 80, la théorie de représentation du discours (DRT [14], [42]) permet de construire la représentation d'un texte en examinant le contenu sémantique dépendant du contexte. L'idée de base est que l'interprétation d'un discours s'appuie sur une représentation sémantique, nommée structure de représentation du discours (DRS [15]), que l'analyse successive des phrases enrichit. Les DRS peuvent être avantageusement exploitées comme représentation pivot pour guider le passage d'énoncés exprimés en langue naturelle vers des spécifications formelles. On les retrouve ainsi dans des travaux de *Fuchs and Schwitter* sur la traduction automatique d'énoncés en clauses Prolog [25]. *Nelken et Francez* propose une approche analogue pour exprimer des spécifications complexes écrites dans un langage naturel relativement libre [43].

Les structures de dépendance sont des descriptions formelles portant sur les liens syntaxiques entre les mots d'un énoncé. Elles sont généralement représentées à l'aide de graphes, où les mots sont associés aux sommets et les fonctions syntaxiques aux arcs. Le projet ARSENAL [44] propose de relier des phrases du langage naturel à des formules logiques temporelles en s'appuyant sur des structures en dépendance. Le cadre méthodologique fournit un processus complet permettant de transposer automatiquement des

14. DRT : Discourse Representation Theory
15. DRS : Discourse Representation Structure

exigences du langage naturel vers des systèmes de transitions exprimés à l'aide du langage de spécification SAL [45] et des spécifications logiques LTL [20]. Les techniques mises en œuvre combinent plusieurs méthodes, dont l'utilisation d'une représentation linguistique intermédiaire. Un processeur sémantique applique un ensemble de règles pour annoter les dépendances. Une représentation intermédiaire est ainsi construite avec des balises de métadonnées, exploitées dans un second temps pour obtenir la formule cible.

Finalement, les représentations sémantiques sont une autre famille de formalismes. Ce type de structures permet de symboliser le sens d'un énoncé tel qu'il pourrait être compris par un locuteur de la langue. Des processus les accompagnent généralement pour en extraire l'information utile. Les grammaires catégorielles, associées au lambda-calcul, peuvent être incluses dans cette catégorie. Il en est de même de la théorie de représentation du discours. D'autres théories ont été développés ces dernières années. Elles visent des objectifs pratiques variés, en mettant davantage l'accent sur les informations sémantiques lexicales. Concrètement, ces structures ciblent particulièrement les rôles sémantiques, le sens des mots, ou encore les entités et leurs relations. Une description plus complète est donnée dans le chapitre 3 pour quelques représentations de sens, telles que les graphes AMR [46], structures sémantiques simples dont le développement est inspiré des banques d'arbres syntaxiques, ou l'UCCA [47], structures à plusieurs niveaux pour l'annotation des distinctions sémantiques.

2.3 Traitement complet

Les techniques de traitement du langage naturel ne peuvent garantir une traduction sans erreur. Ce point de vue est renforcé par la nature intrinsèquement ambiguë des langues naturelles. Nous pouvons toutefois atténuer ce constat par la nature des documents visés, généralement rédigés avec des mots répétitifs, un style restreint et un discours explicite.

D'un autre côté, de nombreuses méthodes ont été développées pour spécifier formellement les comportements attendus d'un système. Celles-ci sont rigoureusement fondées, validées par des preuves mathématiques et des évaluations expérimentales. Malheureusement, suivant l'état actuel des connaissances, ces techniques sont impossibles à mettre en œuvre sur les documents produits par les industriels. La construction de spécifications formelles nécessite une étape de formalisation complexe dont l'automatisation semble inaccessible.

Notre contribution vise à contourner cet obstacle, en combinant des techniques de traitement du langage naturel et des formalismes de haut niveau. Le but est de poser les bases d'un processus entièrement automatisable pour l'analyse formelle de cahiers des charges. Il s'agirait d'être en capacité d'extraire et formaliser automatiquement les exigences d'une spécification, tout en apportant certaines garanties pour contrôler les résultats obtenus.

La figure 4.2 montre les principales étapes d'un système complet partant de documents produits par le concepteur, et aboutissant à une spécification formelle complète, vérifiable et simulable.

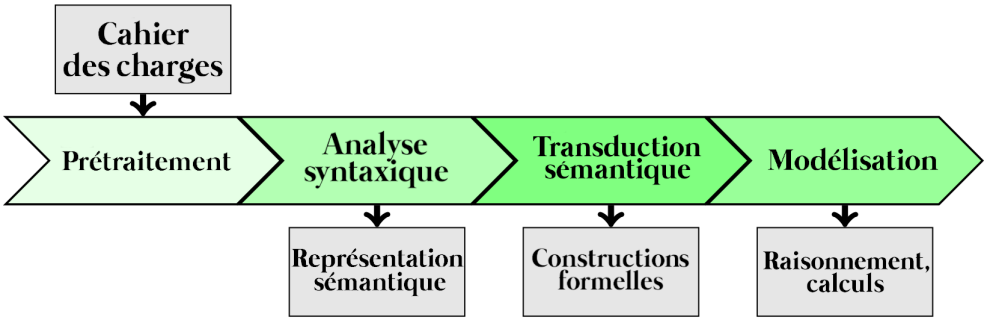


FIGURE 2.3 – Chaîne de traitement

Cette chaîne de traitement peut être observée suivant deux directions opposées. Les deux premières étapes sont tournées vers la source. Partant d'un document contenant une liste de déclarations en langage naturel, elles aboutissent à la construction d'une représentation sémantique pivot pour chaque énoncé. Les deux dernières étapes pointent vers la cible. L'analyse de la représentation pivot permet, en suivant des techniques que nous définirons au chapitre 4, d'extraire et formaliser les exigences à vérifier. Finalement, l'utilisation d'une structure adaptée révèle les incohérences éventuelles et produit, le cas échéant, une modélisation complète de la spécification.

La suite apporte des précisions sur chacune des étapes proposées. Deux structures pivots intermédiaires sont utilisées. La première structure est une représentation de sens, dont nous motivons l'usage dans la section 2.3.2. La seconde structure est un format ad hoc, nommé MPC. Elle est présentée dans la section 2.3.3. Sans être fondamental, son usage favorise la modularité du traitement.

2.3.1 Prétraitements des documents

Il n'y a pas d'enjeu majeur concernant le découpage ou la conversion des documents. Nous considérons donc comme point de départ une liste d'énoncés. Nous ne fixons aucune contrainte à priori sur ces énoncés. Les opérations de prétraitement sont classiques. Elles comprennent le découpage des éléments de chaque phrase en unités syntaxiques (items lexicaux) et l'étiquetage grammatical (POS tagging). Un outil tel que Stanford CoreNLP [48] offre toutes les opérations utiles pour adapter les données aux traitements à venir.

2.3.2 Analyse syntaxico-sémantique des énoncés

Chaque énoncé peut être analysé en suivant une approche linguistique et sémantique. L'objectif de cette étape est de produire une représentation pivot saisissant la signification de ces énoncés. Différentes approches, et différentes structures, peuvent être envisagées, comme nous le verrons au chapitre 3. Dans nos travaux, nous avons choisi d'exploiter en particulier les représentations de sens abstraits (AMR) [46]. Ce formalisme est simple, outillé et adapté. Il est utilisé pour saisir la sémantique au niveau des phrases. Sa structure prend la forme d'un graphe où les nœuds correspondent aux concepts présents dans la phrase. Les arêtes orientées qui relient les nœuds du graphe traduisent les relations entre ces concepts. Cette structure fournit ainsi une représentation formelle préservant le sens de l'énoncé.

Exemple 1 *La figure 2.4 montre la représentation AMR de l'énoncé E_1, "When a ticket has been issued, the entry gate must open", obtenue avec l'analyseur syntaxico-sémantique STOG [49]. Chaque nœud du graphe définit une unité sémantique associée à un concept, tandis que chaque arête orientée définit une relation entre deux concepts. Les concepts sont précisément identifiés en se basant sur une banque de référence (PropBank [50]). Dans cet exemple, le concept "obligate" identifie la modalité déontique introduite dans la phrase avec le verbe "must". Les entités sont désignées avec les concepts "ticket" et "gate". Ces concepts sont reliés aux propositions verbales "issue" et "ticket" par la relation "arg" (le numéro associé apporte une précision supplémentaire sur la relation entre la proposition verbale et l'argument, qui renvoie à une signification précise définie dans la documentation).*

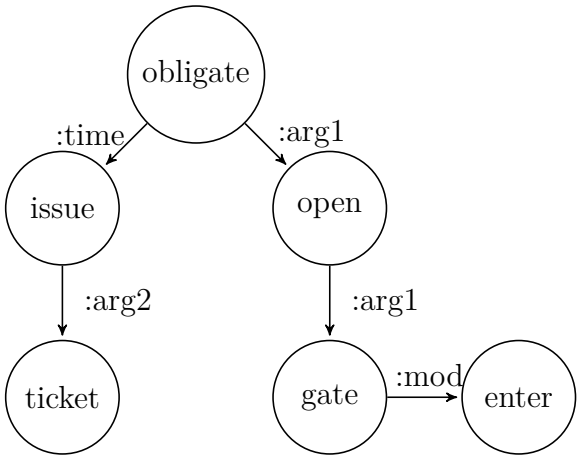


FIGURE 2.4 – Représentation AMR de l'énoncé E_1 obtenue avec l'algorithme STOG [49].

2.3.3 Transformation sémantique de la représentation pivot

L'étape suivante consiste à traiter ces représentations pour les classer et les transformer. Le problème se pose donc en termes différents : ce n'est plus un énoncé du langage naturel qui est traité, mais une représentation formelle de cet énoncé qu'il s'agit d'interpréter.

Nous proposons une méthodologie que nous nommons transduction sémantique. Le principe général est d'appliquer une série de transformations à partir du graphe AMR tout en respectant certaines conditions. Ces conditions prennent la forme de formules logiques traduisant les critères d'application des transformations. Ces dernières consistent en l'application d'opérations qui enrichissent l'interprétation du graphe. Elles permettent de construire des regroupements de nœuds. Associés à de nouveaux concepts, ces ensembles forment, en un certain sens, des abstractions sur différentes parties du graphe.

Les formules logiques et les opérations sont associées dans des schémas que nous appelons schéma de transduction compositionnel (STC). Ces schémas définissent des règles s'appliquant à une interprétation logique de base pour aboutir à une interprétation enrichie, où apparaissent des concepts abstraits. L'un de ces concepts correspond au résultat attendu. Sa présence permet d'obtenir la formalisation attendue, tandis que son absence entraîne le rejet de l'énoncé (qui ne correspond pas à la cible).

Exemple 2 *La figure 2.5 donne un aperçu simplifié des opérations appliquées lors de cette étape. L'analyse du graphe sémantique de l'énoncé E_1 part d'une interprétation brute initiale, constituée d'ensembles formés autour de chaque sommet du graphe. Cette in-*

terprétation est enrichie par application successive de règles dites de transduction. Elles permettent d'aboutir à une représentation de plus haut niveau correspondant à la formalisation attendue en sortie.

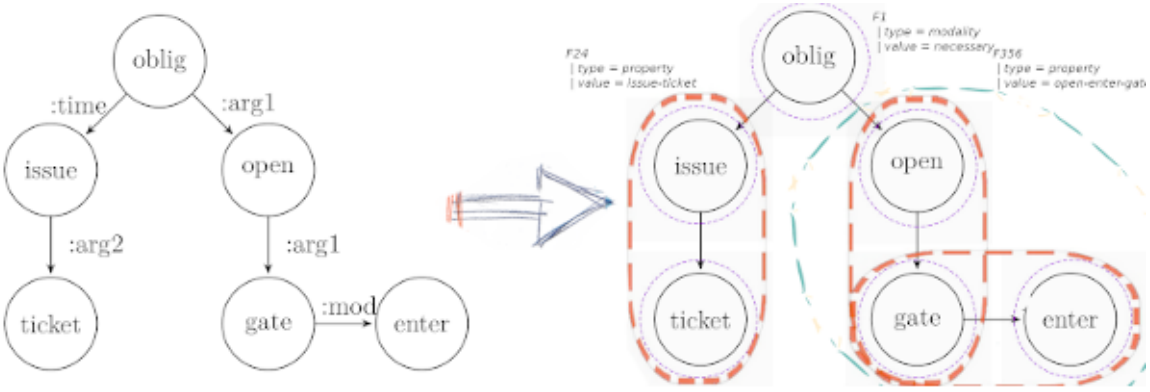


FIGURE 2.5 – Exemple d'analyse par transduction sémantique pour le graphe de l'énoncé E_1

Ces opérations, notées STC, seront précisées et détaillées dans le chapitre 4. Notons que les opérations de transduction sont appliquées jusqu'à atteindre un point fixe. La formalisation est obtenue si l'interprétation finale obtenue est associée au type output. Si ce n'est pas le cas, l'énoncé est rejeté.

2.3.4 Modélisation

Dans le contexte de la conception de systèmes, les théories d'interface permettent de décrire en une seule spécification à la fois le comportement attendu d'un composant et les environnements possibles dans lesquels il peut être exécuté. Les exemples typiques de théories d'interface sont les *automates d'interfaces* [3] et les *interfaces modales* [13, 14, 4].

Exemple 3 (Spécifications modales) *Les spécifications modales sont des systèmes de transitions modaux déterministes [51]. Les systèmes de transitions modaux ont été introduits dans les années 80 [52, 53] pour étudier le raffinement d'actions. Cette opération consiste à remplacer des transitions pour relier les descriptions d'un système à différents niveaux d'abstraction.*

Les spécifications modales peuvent être présentées suivant différentes définitions équivalentes, par exemple sous la forme d'automates. La figure 5.3 montre une spécification modale (à gauche) et deux modèles possibles de cette spécification (à droite), où a et b sont

deux actions (ou événements). Dans cette représentation, la présence d'un arc continu signifie que la transition est obligatoire ("must"), la présence d'un arc en pointillés que la transition est facultative ("may") et l'absence d'arc indique que la transition est interdite dans toute réalisation. La définition d'un modèle pour une spécification modale est déterminée formellement en définissant une relation de satisfaction.

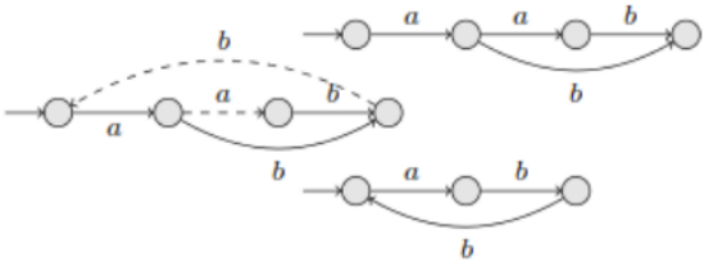


FIGURE 2.6 – Une spécification modale et deux modèles possibles

Les spécifications modales sont dotées de propriétés algébriques particulièrement intéressantes. Ces propriétés sont détaillées dans [13]. On y trouvera en particulier les définitions formelles (et preuves associées) pour la composition ($S_1 \otimes S_2$), la conjonction ($S_1 \wedge S_2$) et le quotient (S_1/S_2) de spécifications modales (avec S_1 et S_2 deux spécifications modales).

Ces formalismes, basés sur la théorie des automates, permettent d'exprimer la variabilité des conceptions envisageables et les incertitudes concernant les environnements possibles d'un composant. Néanmoins, ces théories ne traitent que de la relation entre les entrées et les sorties de composants réactifs en fonction des événements observés. Dans de nombreux cas, il serait souhaitable de relier ces comportements d'entrée/sortie à l'état du composant.

Cette propriété est particulièrement importante pour traduire les exigences des cahiers des charges, comme le montre l'exemple qui suit.

Exemple 4 (Exigences d'un cahier des charges) *Les exigences E_1 et E_2, proposées dans la section 2.1, sont rappelées :*

- *(E_1) When a ticket has been issued, the entry gate must open.*
- *(E_2) The passage of a car is prohibited if the security gate is closed.*

Dans l'exigence E_1, la nécessité (must) d'un comportement (open-gate) est spécifiée pour un état vérifiant une propriété donnée (issue-ticket). De même, l'exigence E_2 interdit (prohibited) un autre évènement (pass-car) pour une autre condition (security-gate-close). Les propriétés d'état, issue-ticket et security-gate-close, ne peuvent pas apparaitre directement dans les formalismes évoqués.

Cette observation est fondamentale. Elle traduit une limite d'expressivité forte en pratique, et restreint de ce fait l'intégration dans un processus automatisé des formalismes cités précédemment. Pour contourner cette difficulté, il est possible de définir indirectement les propriétés d'états, par exemple sous la forme d'un expression rationnelle. Celle-ci peut ainsi caractériser précisément les états vérifiant la propriété, en fonction des événements observés. Mais cette définition formelle, qui dépend de la structure cible, est complexe à mettre en œuvre manuellement. Elle semble à priori impossible à établir automatiquement.

Nous reviendrons sur ce point dans le chapitre 5. Il justifie la proposition de nouvelles structures. Deux contributions émergent de nos travaux : les automates moniteurs, exposés dans le chapitre 7, et les automates déterministes à ensembles d'acceptation propositionnels, développés dans le chapitre 8.

Les automates moniteurs [9, 10] définissent un système de transitions d'états où chaque état est associé à une formule propositionnelle, et où certains états sont signalés comme illégaux. Ce formalisme permet d'observer le comportement d'un système, tandis qu'un modèle définit un système avec différents comportements possibles. Les deux formalismes peuvent être liés par les exécutions du modèle, observées avec le moniteur.

Exemple 5 (Automate moniteur) *La figure 2.7 montre un moniteur qui traduit une règle d'interdiction du passage d'un véhicule si la barrière est fermée.*

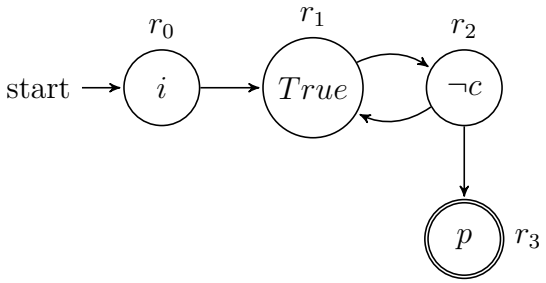


FIGURE 2.7 – Moniteur interdisant le passage d'un véhicule (proposition p : *pass-vehicle*) si la barrière est fermée (proposition c : *close-gate*).

Les automates déterministes à ensembles d'acceptation propositionnels (DPAA [16]) forment une structure un peu moins intuitive, mais avec des propriétés algébriques plus complètes. Ce formalisme combine des propriétés d'état, exprimées sous forme de formules propositionnelles, et des propriétés temporelles simples en temps discret, exprimées sous forme d'ensembles de réalisations possibles. Il prend la forme d'un automate similaire aux automates moniteurs, avec le même fondement propositionnel, mais s'en distingue dans la manière dont s'exprime la variabilité des comportements possibles.

Ces deux formalismes permettent de saisir des exigences comportementales, et notamment les comportements attendus lorsque le système est dans un état particulier. Ils étendent les systèmes de transitions modaux à un cadre propositionnel, où les modèles sont des structures de Kripke plutôt que des systèmes de transitions étiquetés. Une théorie d'interface, reposant sur la définition d'une relation de raffinement et d'opérateurs algébriques (composition, produit et quotient), peut être établie, partiellement pour les automates moniteurs, et complètement pour les DPAA. Une représentation implicite, utilisant des fonctions caractéristiques, permet en outre de limiter la complexité de calcul en temps et en espace. Si l'on se place sur un plan strictement théorique, le niveau d'expressivité des moniteurs ou des DPAA ne dépassent pas celui des spécifications modales. En pratique, la prise en compte directe des propriétés d'états apporte un bénéfice important.

Finalement, il est possible de relier l'une ou l'autre de ces deux structures aux exigences formalisées obtenues dans les étapes précédentes. Il n'y a pas de difficultés majeures, à ce stade, pour dériver les exigences formelles vers ces formalismes. En partant du format MPC, le contexte, défini sous la forme d'une formule logique, peut être directement associé à un état de l'automate. De même, chaque propriété peut caractériser un état interdit (pour les automates moniteurs) ou un ensemble d'acceptation particulier (pour les DPAA), en fonction de la modalité associée. Une autre option consiste à construire directement les automates visés avec des schémas de transduction plus complets.

Nous obtenons ainsi une spécification formelle pour chaque exigence, et la spécification complète par composition de toutes les exigences. Une caractéristique importante de l'opération de composition est que celle-ci aboutit à une structure particulière, n'acceptant aucune réalisation, si deux règles se contredisent. Il est ainsi possible de montrer la consistance d'une spécification en composant toutes les exigences, et de cibler précisément les exigences contradictoires le cas échéant.

16. DPAA : Deterministic Propositional Acceptance Automata

Chapitre 3

Représentations sémantiques pour capturer le sens des énoncés

La modélisation formelle d'une langue peut s'appuyer sur des schémas d'annotations syntaxiques et sémantiques. La Penn Tree Bank [54] est la première banque d'arbres développée à grande échelle. Ce corpus fournit des structures linguistiques pour l'annotation de phrases en langue anglaise, donnant des informations relatives aux parties du discours (POS). L'émergence de ce type de corpus révèle des ensembles de données empiriques de dimensions importantes. Il en résulte des ressources exploitables pour des projets de linguistique informatique, appliqués ou théoriques.

Néanmoins, les nuances sémantiques ne s'y reflètent pas pleinement, ce qui rend ces représentations sensibles aux variations syntaxiques et limite leur valeur pour des applications sémantiques. D'autres structures ont ainsi été considérées, et des projets se sont formés autour de cadres alternatifs. Les projets d'annotation PropBank [50] et NomBank [55] portent sur les propositions verbales et leurs arguments. Le projet FrameNet [56] propose une base lexicale basée sur les cadres sémantiques, une structure conceptuelle décrivant un événement, une relation ou un objet. Les dépendances de type CoNLL [57] fusionnent des dépendances syntaxiques et sémantiques pour obtenir une représentation unifiée. Ces différentes ressources produisent des étiquetages pour les mots d'une phrase, révélant leur rôle sémantique dans la phrase.

Plus récemment, un nombre croissant de travaux portent sur des structures visant à capturer le sens au niveau des phrases. Ces formalismes ont été développés en considérant des perspectives linguistiques variées. Il en résulte des cadres structurels mettant l'accent sur la représentation des informations sémantiques (rôles sémantiques, sens des mots, relations entre les entités). En l'absence de consensus, plusieurs représentations alternatives émergent, avec des caractéristiques inégales. Ainsi, les graphes AMR [46] définissent des structures sémantiques simples, dont le développement est inspiré des banques d'arbres syntaxiques. L'UCCA [47] et la couche TL du corpus PTL [58, 59] présentent des struc-

tures à plusieurs niveaux pour l'annotation des distinctions sémantiques, semblables aux arbres de dépendance. D'autres représentations plus anciennes, telles que les DRT [42, 26], constituent également une option envisageable, au moins sur le plan théorique.

Les structures sémantiques constituent ainsi un intermédiaire intéressant entre l'expression naturelle d'une phrase et son traitement par des méthodes automatiques. Nous proposons d'exploiter ce type de représentations. La finalité est l'étude de documents techniques pour en extraire les données nécessaires à la mise en œuvre de raisonnements formels. Dans un premier temps, notre objectif est de poser les bases d'un processus d'analyse et de transformations sémantiques s'appuyant sur une représentation pivot structurée.

Le propos de ce chapitre s'insère dans un cadre d'analyse générale, visant le traitement automatique d'énoncés en langue naturelle. La section 3.1 donne un aperçu de structures sémantiques pertinentes pour servir en tant que représentations pivots. L'accent est mis sur les graphes AMR, que nous avons exploités plus spécifiquement dans nos travaux. La section 3.2 propose une description complète de ce formalisme, avec plusieurs exemples de représentations pour des énoncés correspondant à des exigences. Finalement, les caractéristiques de ces différents formalismes sont comparées et commentées dans la section 3.3.

3.1 État de l'art

L'usage des représentations sémantiques pour le traitement automatique des langues naturelles évolue rapidement. Ces représentations définissent des structures, indépendantes de tout critère syntaxique, qui reflètent le sens d'un énoncé tel qu'il est compris par un locuteur de la langue. Leur développement répondent à des objectifs pratiques variés, tels que l'analyse automatique d'énoncés ou la synthèse de textes. L'annotation sémantique [60], le résumé automatique de textes [61] ou l'extraction d'informations [62, 63] sont des exemples d'application utilisant efficacement ce type de représentations comme structure pivot.

L'état de l'art comprend de nombreuses propositions qui divergent sur plusieurs aspects [64]. L'annotation cognitive conceptuelle universelle (UCCA, [47]), s'appuyant sur une structure en dépendance à plusieurs niveaux, a été développée pour mettre en évidence les distinctions sémantiques d'un énoncé. La conception est similaire pour la banque d'arbres de Prague (PTL, [58, 59]). La sémantique de décomposition universelle (UDS, [65]) vise à augmenter les données du projet *Universal Dependencies*(UD, [66]) avec des annotations sémantiques robustes et évolutives. Les dépendances universelles profondes

(DUD, [67]) sont également dérivées à partir des arbres de ce projet. Les représentations de sens abstrait (AMR, [46]), à la base d'un corpus alternatif, proposent un cadre de description simple sous la forme de graphes étiquetés. La Groningen Meaning Bank (GMB, [68]) a également pour objectif de proposer un grand corpus de textes anglais annotés avec des représentations sémantiques profondes.

Les caractéristiques de ces structures sont variées. Certaines de ces représentations s'appuient sur un format simple et nécessitent ainsi une expertise moindre. Leurs relations à la syntaxe et leurs liaisons aux mots des phrases s'exercent également à des degrés différents. Ainsi, l'ancrage aux mots d'une phrase donnée est fort pour les structures en dépendances, tandis qu'il est faible, voir inexistant, pour les structures basées sur la logique. Il en résulte des différences importantes dans leurs capacités à s'abstraire des variations conceptuelles et syntaxiques, et à intégrer des phénomènes linguistiques variés.

Il n'y a pas de consensus autour d'une structure sémantique particulière. Des différences majeures existent. Celles-ci se traduisent en termes de formalisme, d'interface syntaxique et de degré d'abstraction. Par exemple, les représentations sans ancrage sont généralement plus souples à l'usage, tandis que les correspondances entre les mots et les éléments sémantiques facilitent l'analyse syntaxique. De fait, cette dispersion traduit des intérêts contradictoires sur les propriétés structurelles attendues, et leurs pertinences en termes d'adéquation linguistique.

Dans la suite, nous commençons par un aperçu sur quelques propositions intéressantes, sans prétendre à l'exhaustivité. L'une des idées défendues dans ce manuscrit est que ce type de structures pourrait être utilement exploité pour l'analyse et la formalisation d'énoncés du langage naturel. A ce stade, il apparaît néanmoins prématuré de privilégier définitivement une représentation spécifique. Même si nous avons adopté le langage AMR dans nos développements, le choix de représentations comme structures pivot reste un point discutable.

3.1.1 Discourse Representation Theory (DRT)

La théorie de la représentation du discours de Kamp (DRT [1], [42, 26]) fournit un premier cadre d'exploration du sens selon une approche sémantique formelle. Elle inclut un niveau de représentations mentales abstraites, appelé structure de représentation du discours (DRS [2]). Cet élément est central pour la DRT. Elle permet de refléter la dépendance

1. DRT : Discourse Representation Theory
2. DRS : Discourse Representation Structure

du sens par rapport au contexte, et confère à la DRT une capacité intrinsèque à traiter le sens au-delà des limites de la phrase.

Une DRS est un objet mathématique qui comporte deux éléments essentiels : un ensemble de référents discursifs représentant des entités, et un ensemble de conditions représentant des concepts liés aux référents. Les conditions peuvent être atomiques ou complexes. La DRS consiste en un prédicat dans le premier cas, avec un nombre approprié de référents discursifs. Les DRS peuvent être combinées par fusion pour produire des conditions complexes.

Exemple 6 *L'idée de base suivie par la DRT est qu'un auditeur construit une représentation mentale du discours au fur et à mesure de son déroulement. Ce principe se reflète dans la construction et l'enrichissement des DRS. Nous reprenons l'exemple classique des phrases "donkey", initialement proposées par le philosophe Peter Geach en 1962 [69]. Ces phrases ont constitué un cadre d'étude important pour différentes théories, dont la DRT.*

L'énoncé "Every farmer who owns a donkey beats it" peut être représenté par la DRS $[x, y : farmer(x), donkey(y), owns(x, y)] \Rightarrow [beat(x, y)]$*. Cette structure intègre deux DRS :* $[x, y : farmer(x), donkey(y), owns(x, y)]$ *et* $[beat(x, y)]$*, qui contiennent deux référentiels discursifs (x et y), et quatre conditions (farmer, donkey, own et beat). Plusieurs notations ont été proposées pour représenter une DRS, dont la notation en boîte utilisée dans la figure 3.1.*

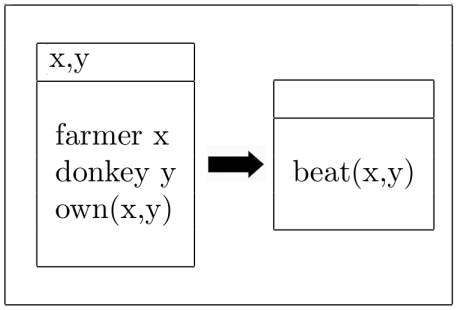


FIGURE 3.1 – DRS de la phrase "Every farmer who owns a donkey beats it"

La DRT est une théorie de la sémantique dynamique. Elle est fondée sur l'idée que la signification réalise, au fil du discours, des modifications successives du contexte discursif. Le lien entre l'interprétation et le contexte y est dual : l'interprétation dépend du contexte tout en créant celui-ci. Cette dualité se traduit par l'imbrication des DRS.

Le langage de représentation de la DRT consiste en une définition récursive de l'ensemble des DRS bien formées. Un modèle sémantique théorique est associé aux éléments de cet ensemble. Ce modèle intègre une procédure de construction qui précise comment étendre une DRS donnée à chaque analyse d'une nouvelle phrase.

Proposée au début des années 80, la théorie de la représentation du discours a été développée dans le but de soutenir l'inférence logique dans les systèmes de traitement basés sur le raisonnement logique. Elle constitue ainsi un cadre de représentation assez ancien, mis en œuvre dans différents projets [70, 71]. Les systèmes exploitant les DRS nécessitent, en complément, une représentation appropriée des composantes sémantiques.

3.1.2 Abstract Meaning Representation (AMR)

Les représentations de sens abstrait (AMR [3], [46]) sont des structures sémantiques simples et complètes. Elles permettent de traduire la signification de toute phrase anglaise sous la forme d'un graphe orienté et étiqueté, où les nœuds et les arêtes sont respectivement associés à des concepts et des rôles sémantiques. Le développement de ce formalisme est inspiré du succès des banques d'arbres syntaxiques, dont la Penn TreeBank est un exemple. Par suite, une ressource d'annotations sémantiques s'est constituée autour des AMR pour associer des phrases anglaises à leur sens logique. Cette banque sémantique a aujourd'hui une taille suffisante pour permettre l'entraînement d'analyseurs sémantiques performants.

Exemple 7 *La figure 3.2 donne la représentation AMR de la phrase "The boy wants to go", issue de l'article fondateur de Banarescu et al ("Abstract Meaning Representation for Sembanking" [46]).*

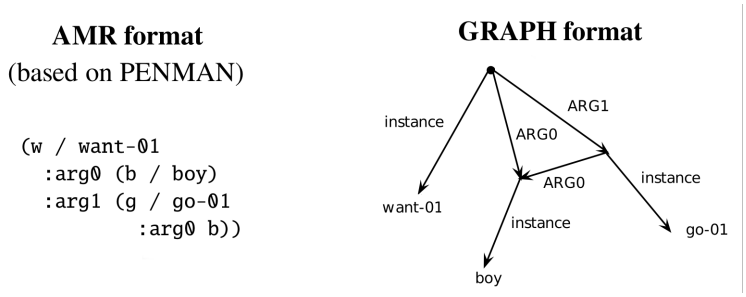


FIGURE 3.2 – AMR de la phrase "The boy wants to go" [46]

3. AMR : Abstract Meaning Representation

L'une de nos hypothèses de travail a porté sur le choix du langage AMR pour nos développements. Ce formalisme apparaît comme adapté à notre démarche. Il présente également l'avantage d'être bien outillé, avec plusieurs analyseurs syntaxico-sémantiques aux performances correctes. Nos travaux ont donc privilégié, en première intention, l'usage de cette structure dont la description est complétée dans la section suivante (section 3.2).

3.1.3 Prague Dependency Treebank (PDT-TL)

Les arbres de dépendance sont des objets mathématiques représentant les liens syntaxiques entre les mots d'une phrase. Les mots sont associés aux sommets de ces structures, tandis que les fonctions syntaxiques étiquettent généralement les arcs reliant les sommets. Par extension, des structures semblables aux arbres de dépendance peuvent offrir des descriptions formelles plus profondes, et conduire à des graphes sémantiques.

Les travaux autour de la banque d'arbres de dépendance de Prague (PDT [4], [58, 59]) suivent cette logique. Cette ressource propose des descriptions linguistiques d'énoncés en langue naturelle (le tchèque). Elle se présente sous la forme d'arbres de dépendance, et contient plusieurs couches descriptives. La plus haute, la couche tectogrammaticale (TL [5]) donne une structure de signification linguistique, introduisant une couche d'abstraction révélant des mots porteurs de sens. Elle définit ainsi un niveau de signification linguistique, et présente une structuration du contenu cognitif. De ce fait, les irrégularités de la forme extérieure des phrases n'apparaissent plus à ce niveau de représentation, qui peut dès lors servir d'interface entre l'analyse linguistique et l'interprétation sémantique.

Exemple 8 *La figure 3.3 est extraite d'un article [72] sur le corpus PCEDT*[6]*, une ressource parallèle tchèque-anglais adaptée à la conception, l'entraînement et l'évaluation de systèmes de traduction automatique. L'arbre tectogrammatical de la phrase "There's a program for women and a science show." est proposée ici. Les nœuds sont associés aux lemmes (ex. : be, and, program), correspondant aux concepts de la phrase, et à des fonctions tectogrammaticales (ex. : PRED, ACT, CONJ), définissant des cadres de valence. Ces derniers révèlent les actants que doit recevoir un concept pour avoir un sens complet. À ce niveau, les irrégularités de la forme extérieure des phrases sont absentes. Cette structure peut donc servir d'interface utile entre l'analyse linguistique et l'interprétation sémantique.*

4. PDT : Prague Dependency Treebank
5. TL : Tectogrammatical Layer
6. PCEDT : Prague Czech-English Dependency Treebank

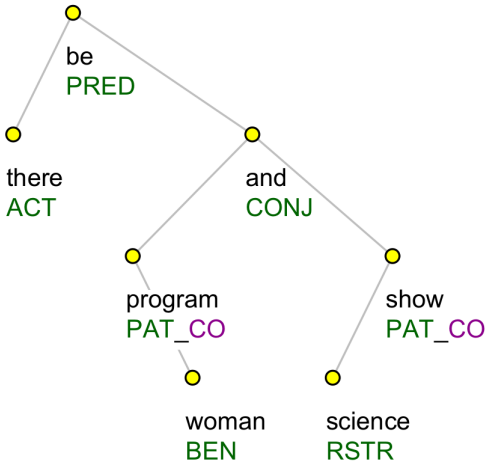


FIGURE 3.3 – Arbre tectogrammatical de la phrase "*There's a program for women and a science show.*" [72]

Les annotations des PDT sont organisées sur quatre niveaux. Le premier niveau est la couche des mots (W-Layer). Les textes sources y sont segmentés en symboles (items), caractérisés dans la couche morphologique (M-Layer). Les phrases sont ensuite représentées sous la forme d'arbres de dépendance, centrés autour du verbe et de sa valence, dans les couches plus profondes, analytique (A-Layer) et tectogrammaticale (T-Layer). Ce dernier niveau propose une structure arborescente reflétant la signification des énoncés.

La conception des PDT est basée sur la théorie de la description générative fonctionnelle (DGF [7], [73]). Ce modèle définit une théorie linguistique relativement complexe, dont plusieurs principes fondamentaux sont repris dans les PDT. Ainsi, les descriptions sont construites sur plusieurs couches successives. De même, l'approche en dépendance de la syntaxe est basée sur la notion de valence [74].

La couche tectogrammaticale (TL) des PDT introduit une couche d'abstraction profonde sur les couches syntaxiques. Elle donne ainsi une structure de sens exploitable, couvrant une grande variété de distinctions fonctionnelles et sémantiques. Elle garde néanmoins une relation étroite avec la syntaxe. Utilisée comme modèle pour plusieurs autres banques d'arbres, les représentations PDT-TL proposent des structures flexibles et génériques, non limitées à une langue particulière. Il est ainsi possible d'adapter ce cadre à la langue anglaise, comme montré dans les travaux de *Čmejrek et al* [72].

7. FGD : Functional Generative Description

3.1.4 Universal Conceptual Cognitive Annotation (UCCA)

L'Annotation Cognitive Conceptuelle Universelle (UCCA [8], [47]) est un cadre à plusieurs niveaux qui permet l'annotation des distinctions sémantiques exprimées par les énoncés linguistiques. Cette représentation s'appuie sur la théorie linguistique de base (BLT [9], [75]), un ensemble de concepts théoriques utilisés pour la description grammaticale des langues et en typologie linguistique.

Les structures sémantiques UCCA sont représentées par des graphes acycliques orientés. Les nœuds du graphe, nommés unités, désignent des terminaux atomiques ou des relations. Les terminaux sont des éléments porteurs de sens (mots ou groupes de mots dans la couche fondamentale), tandis que les relations permettent de caractériser les liens entre les unités. La couche fondamentale de l'UCCA structure les énoncés sous la forme de scènes, qui désignent des actions (ou mouvements) ou des états persistants. La description des éléments d'une scène est réalisée en usant d'un ensemble de catégories.

Exemple 9 *La figure 3.4 donne un exemple d'annotation UCCA pour une scène simple [47]. Le graphe fait apparaître quatre catégories : A (participant), P (processus), E (collaborateur), C (center). La relation principale de la scène pointe vers le verbe kicked. Il y a deux participants, John et his ball. Le concept central du second participant est ball, associée à la caractéristique his.*

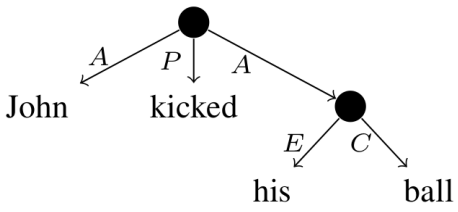


FIGURE 3.4 – Graphe UCCA de la phrase "*John kicked his ball.*" [47]

L'UCCA est une structure à plusieurs niveaux, dont l'extension n'est pas limitée. La première couche est fondamentale. Elle permet de représenter les structures et relations les plus importantes, telles que les verbes, les noms et les adjectifs. Les couches suivantes permettent d'affiner ces représentations.

8. UCCA : Universal Conceptual Cognitive Annotation
9. BLT : Basic Linguistic Theory

L'approche construite autour de l'UCCA n'est pas liée à une langue particulière. La couche fondamentale autorise des extensions ouvertes. De fait, le cadre proposé apparait comme portable, souple et efficacement exploitable. Les schémas sémantiques préservent le sens des énoncés tout en produisant une structure abstraite, relativement insensible aux variations syntaxiques.

3.2 Représentation de sens abstrait

Pour mener à bien nos travaux, nous avons fait le choix d'exploiter les structures AMR en première intention. Ce formalisme définit un langage de représentation à la fois simple et complet. Ainsi, la langue anglaise est convenablement couverte. Les phrases peuvent être représentées sous la forme de graphes AMR. La représentation obtenue présente quelques simplifications. Néanmoins, elle permet de bien faire ressortir les concepts porteurs de sens et leurs relations. Des phénomènes linguistiques essentiels, tels que les modalités ou les connecteurs logiques, sont pris en compte. Des projets d'extension existent pour certaines propriétés ignorées. Le langage AMR offre également l'avantage d'être bien outillée, en particulier pour l'anglais. Les travaux liés à cette représentation aboutissent à une croissance notable de la performance des analyseurs AMR, s'appuyant sur des banques de données de plus en plus complètes.

La suite de cette section complète la présentation de ce formalisme. La structure des graphes AMR est décrite, et les principales caractéristiques sont présentées. Quelques exemples illustrent la représentation d'exigences sous forme d'AMR, mettant en relief certains phénomènes linguistiques. L'accent est en particulier mis sur les modalités déontiques, qui ont une importance critique dans les exigences. Finalement, nous présentons quelques stratégies pour la mise en œuvre d'analyseurs syntaxico-sémantiques produisant des représentations AMR d'énoncés en anglais.

3.2.1 Structure des graphes AMR

La structure des graphes AMR suit la définition traditionnelle des graphes orientés. Les nœuds et les arcs sont étiquetés, l'un des nœuds définissant la racine du graphe. Un concept est associé à chaque nœud, tandis que les relations sémantiques sont spécifiées par les arcs.

Un concept est défini par un mot anglais, un élément issu de la PropBank [50] ou

un mot-clé particulier, permettant d'expliciter certains phénomènes linguistiques. Par exemple, les mots-clés *and* et *or* désignent une conjonction logique, tandis qu'une distance sera caractérisée par le mot-clé *distance-quantity*. Les conventions de la PropBank sont suivies pour la définition des relations sémantiques, notamment pour relier les arguments aux propositions. D'autres relations, porteuses d'un sens plus précis, sont associées à des mots-clés spécifiques.

Exemple 10 *L'exemple de la figure 3.5 montre l'AMR correspondant à la phrase "The machine accepts coins". Les nœuds sont associés aux concepts accept-01, machine et coin. Le premier est une proposition verbale de la PropBank, et qui peut être relié à deux arguments principaux (arg0 et arg1). Ces relations sont représentées par les arcs partant du nœud accept-01. Le premier argument (arg0) est le sujet du verbe, tandis que le second (arg0) est un complément d'objet.*

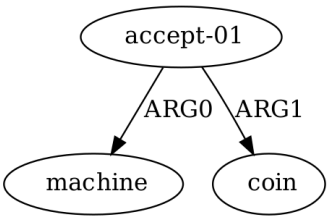


FIGURE 3.5 – AMR de la phrase "The machine accepts coins"

Une documentation complète du langage AMR est donnée dans le document de spécification de référence [76]. En s'appuyant sur les rôles sémantiques de la PropBank, le langage AMR est conçu pour représenter toutes les phrases. La couverture des phénomènes linguistiques est très large. Elle comprend, par exemple, la reconnaissance des entités nommées, la prise en compte des modalités ou la différenciation des énoncés interrogatifs.

Les rôles principaux sont fondés sur la couche sémantique du projet OntoNotes [77]. Ces rôles sont numérotés et spécifiques aux concepts auxquels ils sont liés. Ils sont désignés avec les notations *arg0*, *arg1*, etc. Ils permettent ainsi de relier une proposition centrale à plusieurs arguments, dont l'interprétation est définie dans la banque de référence. En général, l'argument 0 (*arg0*) désigne le sujet de la proposition, tandis que les arguments suivants (*arg1*, *arg2*, ...) désignent les compléments d'objet.

Exemple 11 *La phrase "The passage of a vehicle is prohibited when the gate is closed." peut être représentée par le graphe AMR de la figure 3.6. Cet énoncé définit une exigence interdisant le passage d'un véhicule lorsque la barrière est fermée. Elle fait référence à des entités (les concepts vehicle et gate), un événement (le passage d'un véhicule, qui se traduit par la relation entre les concepts pass-01 et vehicle) et une propriété (la barrière est fermée, qui se traduit par la relation entre les concepts close-13 et gate). La modalité prohibit-01 s'applique sur l'événement, dont la proposition centrale est pass-01. La période d'application est précisée avec la relation time, qui pointe vers la proposition centrale close-13.*

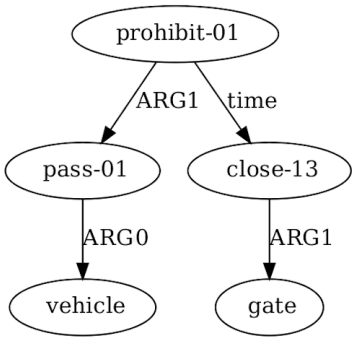


FIGURE 3.6 – AMR de la phrase "The passage of a vehicle is prohibited when the gate is closed."

D'autres phénomènes linguistiques sont spécifiés à l'aide de mots-clés particuliers, précisant un concept ou une relation. Ainsi, le mot-clé *amr-unknown* est utilisé pour indiquer une question. La négation est mise en évidence à l'aide de la relation *polarity*, comme dans la figure 3.7. Les modalités se traduisent par l'usage de concepts spécifiques, tels que *possible-01*, *obligate-01* ou *permit-01*.

Exemple 12 *L'exemple suivant représente la phrase "It is not possible to ask for a drink in stand-by mode.". Les concepts sémantiques sont associés aux nœuds. En particulier, le concept possible-01 spécifie une modalité qui s'applique sur un événement dont la proposition centrale est request-01. Une négation est associée à la modalité. Elle est représentée par la relation polarity.*

La définition du langage AMR inclut également des rôles sémantiques complémentaires, permettant de préciser le sens d'un concept. Par exemple, le rôle *duration* permet

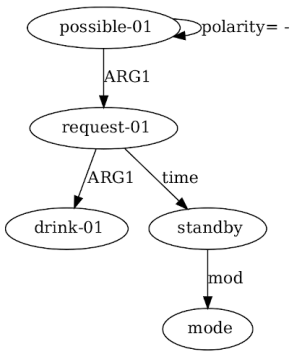


FIGURE 3.7 – AMR de la phrase "It is not possible to ask for a drink in stand-by mode."

de définir la durée d'un événement. Le but et la cause sont spécifiés à l'aide des rôles *purpose* et *cause*, la condition à l'aide du mot-clé *condition*, etc. Dans l'exemple suivant, le rôle *time* précise la période où la barrière peut s'ouvrir.

Exemple 13 *L'AMR de la figure 3.8 représente la phrase "The entry gate may open if a ticket has been issued.".*

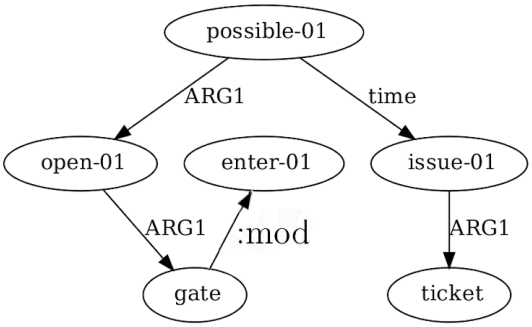


FIGURE 3.8 – AMR de la phrase "The entry gate may open if a ticket has been issued."

3.2.2 Analyse syntaxico-sémantique

La banque de sens AMR fournit un corpus de bonne taille, dont la couverture par des algorithmes d'analyse représente un défi important. Ainsi, les analyseurs syntaxico-sémantiques pour l'AMR visent la correspondance de phrases avec leurs représentations formelles. La recherche d'algorithmes performants s'est traduite par de nombreux travaux, explorant des approches variées.

L'état de l'art dévoile quatre grandes tendances. Les premiers analyseurs efficaces étaient basés sur une tâche d'alignement entre les mots et les concepts, en s'appuyant sur des graphes de dépendance. Une démarche itérative sur les nœuds des arbres en dépendance constitue une alternative intéressante. Des techniques d'induction grammaticale ont également été explorées, certaines approches posant le problème de l'analyse comme un problème de traduction automatique. Plus récemment, l'exploitation de modèles neuronaux basés sur l'attention, reliant des séquences à des graphes, a permis d'améliorer la justesse sémantique des représentations produites.

Toutes ces approches peuvent être évaluées à l'aide d'une mesure de précision de type F-Score. Cette mesure est définie comme la moyenne harmonique de la précision et du rappel d'un modèle sur un ensemble de données. Largement utilisée pour évaluer les algorithmes de classification, elle combine ainsi l'estimation de la précision et de la fiabilité d'un système d'analyse. L'appréciation d'une structure sémantique implique néanmoins quelques spécificités. En particulier, l'estimation brute d'un résultat, sous forme de rejet ou d'acceptation, ne permet pas de produire des éléments comparatifs pertinents. Le score Smatch a été proposé pour remédier à ce défaut [78]. Cette métrique évalue le degré de chevauchement entre deux structures sémantiques, et permet de comparer les résultats d'algorithmes à partir d'une base de référence. Elle s'est imposée comme mesure de référence pour les analyseurs sémantiques. Les études récentes mettent en lumière une progression régulière des résultats obtenus, avec des scores dépassant les 75 % sur les derniers corpus. Des résultats assez proches ont été obtenus pour différents algorithmes, suivant des techniques variées. Ce champ d'étude reste relativement actif. Les tendances n'apparaissent pas définitives.

La suite de cette section est un parcours des principales approches envisageables.

Alignement entre les mots et les concepts. Un premier analyseur AMR, introduit par *Flanigan et al.* en 2014 [79], produit une analyse syntaxique basée sur l'alignement entre les mots et les concepts. L'identification des concepts est d'abord traitée comme une tâche d'étiquetage de séquences. Les relations sont ensuite identifiées à l'aide de graphes de dépendance. Une approche similaire a été utilisée par *Zhou et al* [80], où l'identification des concepts et des relations est effectuée conjointement, et *Liu et al* [81], où les alignements sont utilisés comme variables latentes d'un modèle probabiliste conjoint, avec une amélioration substantielle des résultats obtenus.

Analyse en dépendances. Les graphes d'analyse en dépendance peuvent également être exploités par un processus d'analyse itératif fondé sur l'analyse des transitions. Cette approche consiste à transformer progressivement les analyses de dépendance en structures AMR, avec un traitement sur les nœuds des arbres de dépendance. L'algorithme CAMR [82] a été la première mise en œuvre de cette stratégie. Il a été suivi par de nombreux travaux analogues obtenant de bons résultats, par exemple *Puzikov et al* [83] ou *Lyu et al* [81].

Induction grammaticale. Les techniques d'induction basées sur les grammaires exploitent des ressources sémantiques pour convertir des formules logiques en graphes AMR. Les travaux d'*Artzi et al* [84] combinent ainsi une analyse syntaxique basée sur les CCG [17] pour traiter les aspects compositionnels du sens, et un graphe factoriel pour modéliser les phénomènes non compositionnels. Une approche similaire a été explorée par *Peng et al* [85], tandis que le problème d'analyse a été traité par *Pust et al* [86] comme un problème de traduction automatique.

Modèle basé sur l'attention. Des modèles basés sur l'attention peuvent être exploités par des algorithmes de prédiction reliant des séquences à des graphes. L'algorithme Sequence-to-graph proposé par *Zhang et al* [49] apparaît comme particulièrement efficace. Il s'appuie sur un apprentissage limité d'un mécanisme de copie côté source, et ne fait qu'un usage limité de ressources sémantiques externes. Le processus d'apprentissage est ainsi relativement efficace, même avec une quantité restreinte de données d'entraînement.

3.3 Discussion

Il n'est pas possible, à l'heure actuelle, de répondre à de nombreuses questions sémantiques sur des textes, avec une forte fiabilité, sans recourir à des annotations manuelles. Les banques de données associées aux structures sémantiques sont une réponse à cette limite, qu'elles devraient aider à dépasser. Elles forment des ressources précieuses pour l'entraînement d'algorithmes performants, dont la conception révèle un champs de recherche actif.

Ce manuscrit s'inscrit dans ce cadre. Notre objet n'est pas, cependant, de proposer une représentation alternative, mais de montrer comment exploiter efficacement les représentations existantes pour des tâches d'analyse et de formalisation d'énoncés du langage

naturel. Les principes de transduction sémantique, exposés dans le prochain chapitre, incarnent notre principale contribution dans ce domaine. Ils s'appliquent sur toute structure à base de graphes, telles que les graphes AMR, les structures UCCA ou les arbres de la couche tectogrammaticale de la PDT.

Ce chapitre a permis de parcourir les propriétés de quelques structures, pertinentes pour symboliser le sens d'énoncés en s'affranchissant des contraintes syntaxiques. Sans viser l'exhaustivité, nous avons cherché à montrer la diversité et le dynamisme des projets construits autour de ce type de données. L'une des questions qui se posent naturellement est celle du choix de la représentation. Nous avons mis l'accent sur les graphes AMR. Ces représentations apparaissent simples, relativement complètes et bien outillées. Comme nous le verrons, nos expérimentations montrent que l'analyse de ce type de graphes par transduction sémantique est efficace. Néanmoins, à ce stade, aucune conclusion définitive ne peut être formulée. Il serait ainsi intéressant de comparer l'exploitation de représentations alternatives.

Chapitre 4

Principes d'analyse par transduction sémantique

Les représentations sémantiques forment un pont entre l'expression naturelle d'un énoncé et son traitement par une méthode automatique. Nous avons vu, au chapitre précédent, différentes structures pertinentes. À ce stade, la question posée est celle de l'exploitation de telles structures. Précisément, nous proposons de définir une méthodologie de traitement, applicable aux représentations à base de graphes, et permettant d'obtenir des déclarations formelles respectant un format précis.

Notre contribution prend la forme d'un processus d'analyse basé sur le principe de transduction sémantique. L'idée est d'appliquer une série de transformations sur l'interprétation d'un graphe sémantique tout en respectant certaines conditions. Celles-ci traduisent les critères d'application des transformations. Elles s'expriment à l'aide de formules logiques. Les transformations consistent en l'application d'opérations qui enrichissent l'interprétation du graphe analysé, pour produire, in fine, la déclaration attendue.

Ces principes peuvent s'appliquer à toute structure pouvant être plongée dans un graphe étiqueté. La notion de graphe sémantique élémentaire est introduite comme objet du traitement d'analyse, permettant ainsi de s'affranchir des contraintes particulières de la représentation sémantique utilisée. Cette condition est tout à fait raisonnable, et vérifiée, de fait, par la plupart des représentations envisagées.

Une autre notion est également introduite pour préciser l'interprétation du graphe sémantique. Librement inspirés de la théorie des filtres [87], les filets sémantiques permettent de construire des regroupements sur les sommets d'un graphe. Ces regroupements identifient des ensembles de sommets typés et étiquetés. Ils traduisent, en quelque sorte, des interprétations partielles du graphe, qu'il est possible de composer pour obtenir l'interprétation attendue, à savoir une interprétation associée à la déclaration formelle cible.

Ces deux notions sont illustrées par la figure 4.1, qui montre un graphe et quelques filets (en pointillés). Ceux-ci sont initialement associés à chacun des nœuds du graphe analysé

(un filet par nœud). Ils sont ensuite engendrés par composition des regroupements de base, et par itération du processus sur les nouveaux regroupements obtenus. Des opérateurs sont introduits à cet effet, et des schémas, nommés schémas de transduction, sont définis pour contrôler ces opérations en associant des formules logiques aux opérateurs.

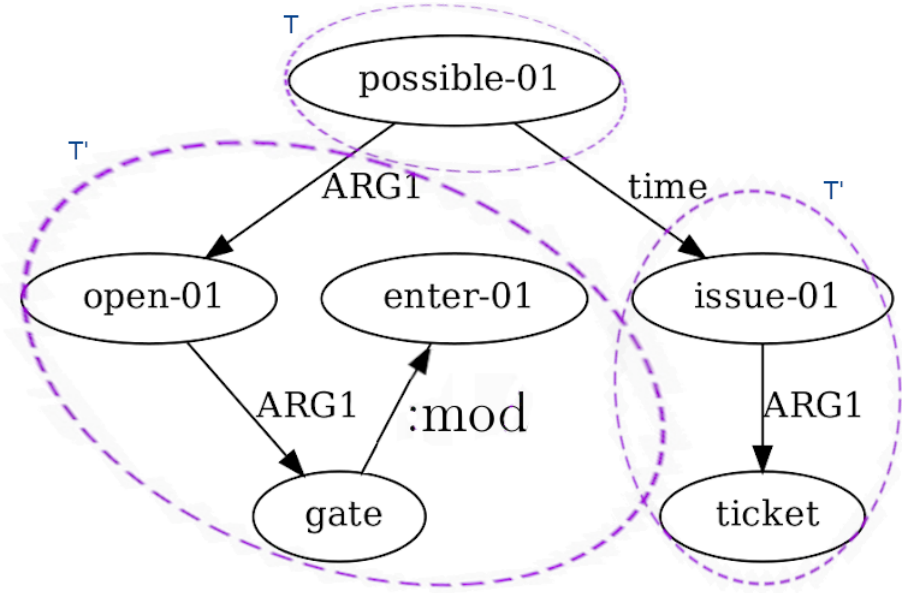


FIGURE 4.1 – Exemple d'un graphe sémantique, avec trois filets quelconques. Le graphe est une représentation de la phrase "The entry gate may open if a ticket has been issued."

Tous ces concepts sont définis plus loin. Ils sont au cœur de l'algorithme d'analyse proposé. Ils permettent ainsi d'établir une méthodologie simple, efficace et facilement adaptable. De fait, l'ensemble du processus est transparent. Il est donc possible de tracer complètement les étapes successives de transformation. De plus, aucune perte n'étant observée dans le traitement des phénomènes linguistiques, leur couverture par transduction sémantique est à priori équivalente à celle de la représentation sémantique utilisée. L'application de ces principes ouvre ainsi la voie à des systèmes lisibles et performants, capables d'interpréter finement des énoncés exprimés en langage naturel.

Cette démarche peut être replacée dans le *workflow* considéré pour l'analyse des exigences d'un cahier des charges, dont nous rappelons les principales étapes avec la figure 4.2. Ce cadre peut servir de base pour un système complet partant de documents produits par le concepteur, et aboutissant à une spécification formelle vérifiable par une méthode de raisonnement adaptée. La phase de transduction sémantique est la troisième étape de ce diagramme.

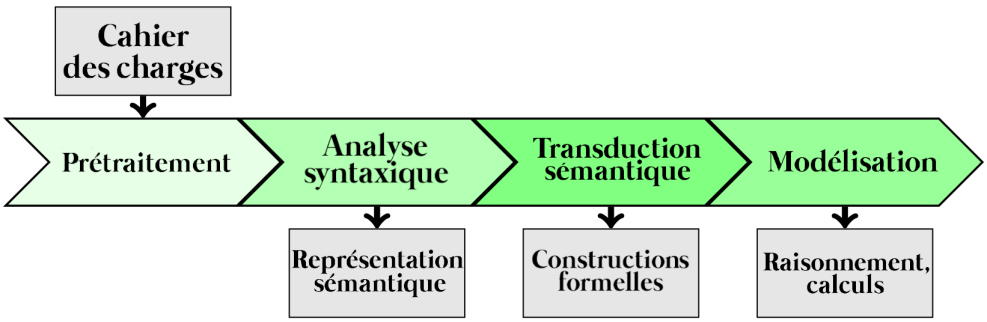


FIGURE 4.2 – *workflow* pour l'analyse des exigences d'un cahier des charges

Nous proposons de traiter les exemples de ce chapitre dans ce contexte, l'objectif étant de produire une déclaration formelle respectant un format donné. Un format ad hoc, le format MPC, est utilisé comme cible. Il permet de définir une relation formelle entre un contexte et une propriété, cette dernière étant associée à une modalité. Une telle déclaration peut être exploitée pour produire une spécification formelle, comme nous le verrons dans les chapitres 5 à 8.

La mise en œuvre de ces principes a permis d'observer une large couverture d'énoncés exprimés en anglais. Il n'y a pas de contraintes particulières sur les propositions verbales ou les noms. Les schémas de transduction sont définis en s'appuyant sur la connaissance de la structure pivot utilisée. Nos expériences ont ciblé quelques phénomènes linguistiques particuliers, essentiels pour l'expression d'une exigence. Les modalités déontiques et temporelles ont ainsi pu être pris en compte, tout comme la négation et les connecteurs logiques. Les résultats obtenus montrent une grande efficacité du processus, dont l'extension rend envisageable une couverture complète de la langue anglaise.

Le reste de ce chapitre s'organise comme suit. La section 4.1 expose les définitions utiles, en précisant les structures manipulées par le processus d'analyse proposé. Les paramètres de transduction sont décrits dans la section 4.2. Ces paramètres sont à la base de l'algorithme d'analyse, dont les principes sont détaillés dans la section 4.3. Finalement, la section 4.4 permet d'évaluer la mise en œuvre du processus. La méthodologie proposée a été implémentée, et évaluée sur un corpus couvrant différents phénomènes linguistiques. Les performances du processus ont pu être démontrées, même si les résultats obtenus restent à confirmer sur des données plus volumineuses, complexes et variées.

4.1 Modèle sémantique

Comme nous l'avons vu au chapitre précédent, il est possible d'associer aux énoncés d'une langue naturelle des structures qui en reflètent le sens, tout en s'affranchissant des critères syntaxiques. Nous observons l'émergence de formalismes variés, traduisant des attentes variables sur les propriétés attendues. Cette diversité se traduit par de nombreux travaux, et un certain dynamisme dans ce domaine d'étude. Il en résulte une hétérogénéité de propositions, sans consensus fort autour d'un formalisme unique.

Nous cherchons à définir un processus permettant d'interpréter de telles structures. Pour gagner en souplesse, nous proposons de nous affranchir du choix de représentation en définissant une structure élémentaire, dans laquelle la représentation utilisée pourra être projetée. Cette projection est réalisable à l'aide des techniques classiques de transformation de structures. Dans la suite, nous définissons la notion de graphe sémantique élémentaire, et nous considérons cette structure comme l'objet du traitement d'analyse.

Un autre concept est également introduit, la notion de filet sémantique, pour contrôler l'interprétation du graphe sémantique. Un filet sémantique permet de capturer, en un certain sens, des sommets d'un graphe. Ce concept permet de construire différents regroupements de sommets, et d'y associer un type et une valeur. De manière analogue, un filet sémantique peut être vu comme un sous-graphe typé.

Nous verrons plus loin comment ces objets peuvent être manipulés. Intuitivement, l'algorithme d'analyse part du graphe de base et d'un ensemble de filets associés à chacun des sommets. Ces filets traduisent, en quelque sorte, une interprétation conceptuelle brute du graphe. De nouveaux filets sont ensuite formés par combinaison des filets existants. L'émergence de ces filets permet d'affiner l'interprétation du graphe, pour aboutir *in fine* à la déclaration formelle attendue.

Dans un premier temps, nous proposons d'expliciter les concepts de graphe sémantique élémentaire et de filets sémantiques. Quelques exemples sont donnés pour illustrer les définitions.

4.1.1 Graphe sémantique élémentaire

Un graphe sémantique élémentaire est un graphe dont les sommets et les arcs sont étiquetés. Deux alphabets sont introduits, Σ_P pour les propositions associées aux sommets, et Σ_R pour les relations associées aux arcs.

Définition 1 *Soit $\Sigma = \Sigma_P \cup \Sigma_R$ un alphabet de propositions et de relations. Un graphe sémantique sur Σ est un graphe $\mathcal{G} = (S, A, \kappa, \eta)$ tel que :*

- *S est un ensemble de sommets ;*
- *$A \subseteq S \times S$ est un ensemble d'arcs ;*
- *$\kappa \subseteq S \times \Sigma_P$ est un ensemble de propositions sur les sommets ;*
- *$\eta \subseteq A \times \Sigma_R$ est un ensemble de relations sur les arcs.*

Un tel graphe correspond pleinement à un graphe AMR, avec un étiquetage libre. Cela permet de s'affranchir des contraintes particulières d'un graphe AMR pour la suite du traitement, qui s'applique à toute structure sémantique pouvant se plonger dans un graphe.

Exemple 14 *La figure 4.3 montre deux exemples de transposition correspondant à des structures du chapitre précédent. Le premier graphe, à gauche, représente la phrase "The boy wants to go." en partant d'une structure AMR. Le graphe AMR a été initialement proposé dans l'article fondateur de Laura Banarescu et al. [46]. Les étiquettes des nœuds et des arcs reflètent respectivement des concepts et des rôles sémantiques. Le second graphe est une transposition de la phrase "John kicked his ball." en partant d'un graphe UCCA. Le graphe UCCA est issu de l'article de Omri Abend et Ari Rappoport [47]. Les étiquettes associées aux arcs correspondent à des catégories relationnelles (A pour participant, P pour processus, E pour collaborateur, C pour center). La relation principale pointe vers le verbe "kicked" d'un scène où deux participants apparaissent ("John" et "his ball").*

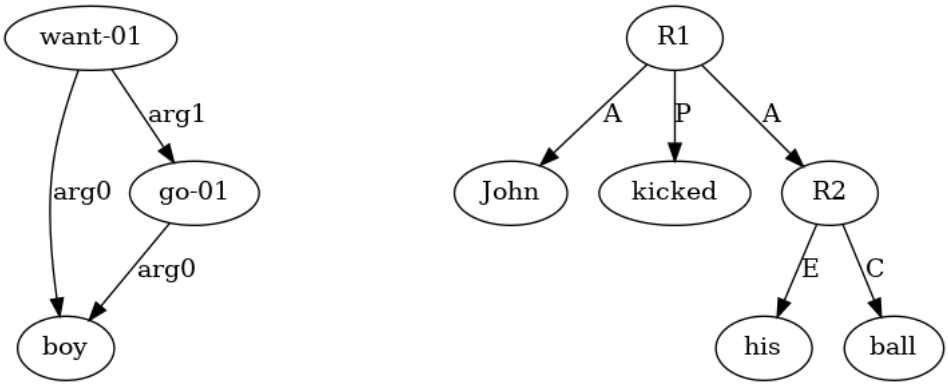


FIGURE 4.3 – Graphes sémantiques des phrases "*The boy wants to go.*" (à gauche) et "*John kicked his ball.*" (à droite), respectivement basés sur un graphe AMR et un graphe UCCA.

Adaptation du graphe initial. Il peut s'avérer nécessaire d'ajuster le graphe initial pour respecter le format de la définition 1. En effet, il est entendu que l'alphabet Σ contient un ensemble de labels correspondant aux propositions (ou concepts) associées aux nœuds du graphe, et aux relations associées aux arcs. Les nœuds et les arcs du graphe traité sont donc associés à un unique label. Ainsi, quelques ajustements du graphe AMR peuvent être utiles. Pour différencier explicitement l'instance d'un concept par rapport au concept lui-même, le graphe pourra être étendu pour faire apparaître deux nœuds distincts, reliés par la relation "instance". Par exemple, l'instance "b" peut être différenciée du concept "boy" par deux nœuds respectivement associés aux propositions "b" et "boy", et reliés par la relation "instance". De même, la négation peut être représentée dans le graphe élémentaire en introduisant un nœud supplémentaire, associé à la proposition "-", et pointé par la relation "polarity".

4.1.2 Filets sémantiques

Librement inspirée de la théorie des filtres[87], la notion de filet sémantique permet de construire des regroupements sur les sommets d'un graphe sémantique. Considérant l'ensemble $\mathcal{S}$ des sommets du graphe, les filets sémantiques se rapportent à l'ensemble des parties de $\mathcal{S}$, en les reliant à des attributs particuliers (type et valeur).

Concrètement, nous cherchons à définir un nouvel objet mathématique qui vise à enrichir un graphe étiqueté, sans le modifier. Cet objet est introduit avec la définition 2. Les premiers filets que l'on ajoute se posent sur chaque sommet du graphe. On y associe un type et une valeur, qui correspondent initialement aux propriétés du sommet ainsi "capturé". Ces notions complémentaires autoriseront l'interprétation et la manipulation des filets, de sorte qu'il deviendra possible de faire émerger de nouveaux filets par composition.

Définition 2 *Filet sémantique $\mathcal{F}$.*
Soit $\mathcal{G} = (S, A, \kappa, \eta)$ un graphe sémantique, avec S l'ensemble des sommets du graphe. Soit $\mathcal{T}$ un ensemble ordonné de types, et $\mathcal{V}$ un ensemble ordonné de valeurs. Un filet sémantique sur $\mathcal{G}$ est une structure $\mathfrak{f}_\mathcal{G} = (X, x_0, t, v)$ telle que :

- *$X \subseteq S$ est un sous-ensemble de S ;*
- *$x_0 \in X$ est la racine du filet ;*
- *$t \in \mathcal{T}$ est un type.*
- *$v \in \mathcal{V}$ est une valeur.*

Typage. Un ensemble ordonné de types est introduit pour spécifier différentes catégories de filets. Les types seront notamment exploités pour contrôler les opérations à appliquer sur un filtre. L'ensemble de types est supposé muni d'une relation d'ordre, permettant ainsi d'établir plusieurs niveaux d'analyse.

Exemple 15 *Dans la suite, nous distinguerons notamment les types init, modality, property, entity, context et mpc, parfois sous leurs formes abrégées i, m, p, e, c et mpc. Cet ensemble peut être complété pour exprimer des niveaux d'analyse différents sur une même catégorie de données. Par exemple, une propriété pourra être associée au type p1 à l'issue d'une première analyse, p2 si elle est obtenue à partir d'une autre propriété, et p3 si sa polarité est négative. La relation d'ordre est définie en considérant* $i < m$, m, p, e *incomparable,* $m < c$ *et* $c < mpc$. *Les types complétés d'un indice reprennent l'ordre naturel des entiers.*

Valeurs. L'ensemble de valeurs $\mathcal{V}$ pourra être muni d'opérateurs permettant de modifier les valeurs, ce que nous verrons avec les définitions 7 et 8. Cet ensemble est également supposé muni d'une relation d'ordre pour consolider la définition de la relation d'ordre sur les filets (défintion 3).

Notations. La notation $\mathcal{F}_{(\mathcal{G},\mathcal{T})}$ désigne un ensemble de filets sémantiques sur le graphe $\mathcal{G}$ et l'ensemble $\mathcal{T}$. Les notations $\mathfrak{f}$ et $\mathcal{F}$ sont utilisées s'il n'y a pas d'ambiguïté sur le graphe et l'ensemble de types, pour dénoter respectivement un filet sémantique ou un ensemble de filets. La classe de tous les graphes et de tous les ensembles de filets sémantiques sur les graphes est notée $\mathcal{C}_{\mathcal{GF}}$.

Exemple 16 *La figure 4.4 reprend l'exemple précédent en associant quelques filets sémantiques aux graphes proposés. Les filets proposés sont associés au type* entity. *Sur le graphe de gauche, l'entité* boy *est centrée sur le concept correspondant. Dans le graphe de droite, l'entité* his-ball *englobe trois nœuds, dont les deux nœuds terminaux révélant les concepts* his *et* ball.

Attributs. Dans les exemples proposés ci-dessus, les filets sémantiques sont présentés avec un attribut unique, leur type. La définition 2 désigne deux attributs : *type* et *valeur*. Ce deuxième attribut permet d'associer, par exemple, une désignation formelle à chaque filet. Nous verrons plus loin la définition d'un opérateur permettant de composer différents

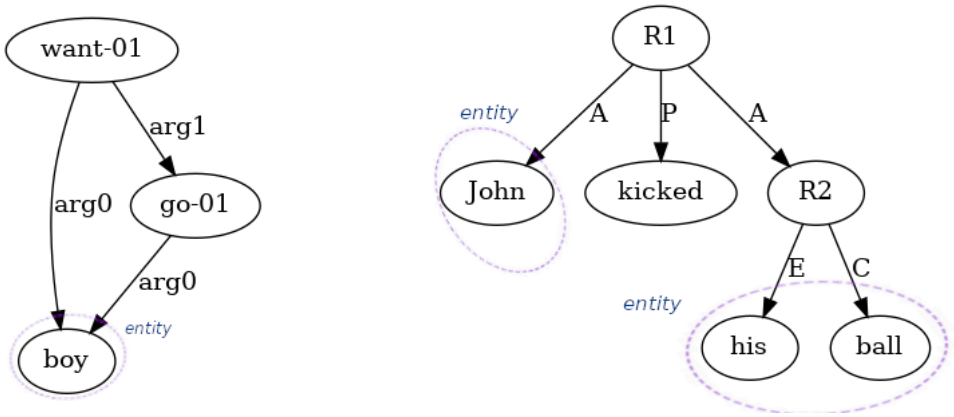


FIGURE 4.4 – Graphes sémantiques et quelques filets sémantiques pour les phrases "*The boy wants to go.*" (à gauche) et "*John kicked his ball.*" (à droite), respectivement basés sur un graphe AMR et un graphe UCCA.

filets, en y associant de nouvelles valeurs. Par extension, des attributs complémentaires peuvent être utilisés en suivant les mêmes principes.

Propriétés et relations. Les propriétés d'un filet peuvent être considérées en prenant en compte son type ou sa valeur, qui représente en quelque sorte l'ensemble des propriétés vérifiables par un filet sémantique. Ce point permet d'étendre aux filets sémantiques la notion de propriétés des nœuds d'un graphe. Les relations entre les nœuds d'un graphe sont transposées aux filets sémantiques de la même manière.

Par extension, cette notion est généralisée en considérant des propriétés associées à un ensemble de types ou de valeurs (inclus dans $\mathcal{T}$ ou $\mathcal{V}$). Ainsi, considérant une propriété p associée à un ensemble de concepts C_p, la propriété p est vérifiée par $\mathfrak{f} = (X, x_0, t, v)$ si $t \in C_p$ ou $v \in C_p$. Par exemple, la propriété *modality* peut être associée à l'ensemble des modalités $C_{modality} = \{may, must, can, \ldots\}$. Elle est vérifiée par le filet $\mathfrak{f}$ si la valeur de ce filet correspond à l'un des éléments de l'ensemble $C_{modality}$. Les propriétés d'un filet peuvent également être examinées en fonction des attributs complémentaires pour un filet dont la définition serait étendue. Les notations $p(\mathfrak{f})_t$ ou $p(\mathfrak{f})_v$ peuvent être utilisées pour désigner une propriété p vérifiée selon t ou selon v. Ces deux extensions ne sont pas fondamentales, mais elles permettent de simplifier la définition des paramètres de transduction que nous verrons plus loin.

Relation d'ordre L'ensemble des filets sémantiques sur un graphe donné peut être muni d'une relation d'ordre s'appuyant sur la relation d'ordre des types et l'inclusion

des ensembles de sommets. Pour un type de filets donné, la relation d'ordre reflète des différences en terme de finesse d'interprétation, en fonction des sommets couverts par les filets.

Définition 3 *Relation d'ordre*
Soit $\mathcal{F}_{(\mathcal{G},\mathcal{T})}$ un ensemble de filets sémantiques sur $\mathcal{G}$. L'ensemble $\mathcal{F}_{(\mathcal{G},\mathcal{T})}$ est muni d'une relation d'ordre $\leq$ telle que, pour tout $\mathfrak{f}_i = (X_i, x_{0,i}, t_i) \in \mathcal{F}_{(\mathcal{G},\mathcal{T})}$, $\mathfrak{f}_1 \leq \mathfrak{f}_2$ si, et seulement si, l'une des trois conditions suivantes est vérifiée :

- $t_1 < t_2$,
- $t_1 = t_2$ *et* $X_1 \subseteq X_2$,
- $t_1 = t_2$, $X_1 = X_2$ *et* $v_1 \leq v_2$.

Il s'agit d'une relation d'ordre partiel. Deux filets sont incomparables si leurs ensembles de sommets X, leurs types ou leurs valeurs sont incomparables. Il est possible que deux filets associés au même ensemble de sommets coexistent, avec des types différents, traduisant des interprétations différentes de ces sommets et de leurs relations.

Les ensembles de types, de sommets et de valeurs étant finis, il existe nécessairement un ou plusieurs filets maximaux (filets des types et valeurs maximaux incluant l'ensemble des sommets).

Exemple 17 *Le graphe sémantique représentant la phrase "The boy wants to go." peut être complété en ajoutant les filets $\mathfrak{f}_2$ et $\mathfrak{f}_3$ de type property, comme illustré sur la figure 4.5. Ces deux filets peuvent être considérés comme plus fins que le filet $\mathfrak{f}_1$ associé au seul sommet du concept boy, de même pour le filet $\mathfrak{f}_3$ par rapport au filet $\mathfrak{f}_2$. L'interprétation peut être considérée comme plus fine et complète pour le filet $\mathfrak{f}_3$ en comparaison au filet $\mathfrak{f}_2$.*

Poids d'un ensemble de filets sémantiques La notion de poids d'un ensemble de filets sémantiques est introduite pour donner une mesure quantitative du volume d'interprétations contenu dans les filets sémantiques. La richesse d'un ensemble de filets s'exprime dans le nombre de types introduits, traduisant ainsi la variété des interprétations incluses, et dans la finesse des filets pour chacun de ces types.

Définition 4 *Poids d'un ensemble de filets*
Le poids d'un ensemble de filets $\mathcal{F}$ pour un type donné t, noté $||\mathcal{F}||_t$, est égal au cardinal

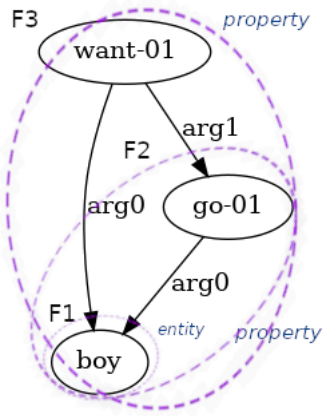


FIGURE 4.5 – Compléments sur le graphe sémantique de la phrase "*The boy wants to go.*"

de l'ensemble des sommets couverts par les filets de type t : $||\mathcal{F}||_t = Card(\bigcup_{\forall \mathfrak{f}=\{X,x_0,t,v\}} X)$. *Le poids total d'un ensemble de filets* $\mathcal{F}$*, noté* $||\mathcal{F}||$*, est défini comme la somme des poids pour chaque type de filets contenus dans l'ensemble* $\mathcal{F}$ *:* $||\mathcal{F}|| = \sum_{\forall t \in T} ||\mathcal{F}||$.

Exemple 18 *Considérons l'ensemble des filets sémantiques* $\mathcal{F} = \{\mathfrak{f}_1, \mathfrak{f}_2, \mathfrak{f}_3\}$ *du graphe de la figure 4.5. Le poids de cet ensemble est 1 pour le type* entity *et 3 pour le type* property*, tandis que le poids total vaut 4. Le poids total ne serait que de 3 en l'absence du filet* $\mathfrak{f}_1$*, traduisant l'absence d'interprétation pour le type* entity*. Il passerait à 3 en l'absence du filet* $\mathfrak{f}_3$*, traduisant une finesse moindre de l'interprétation pour le type* property*. En revanche, l'absence du filet* $\mathfrak{f}_2$ *serait sans conséquence sur le poids de l'ensemble, le filet restant* $\mathfrak{f}_3$ *du type* property *étant plus fin que le filet retiré.*

4.2 Opération de transduction

L'analyse par transduction sémantique opère sur une structure associant un graphe et un ensemble de filets sémantiques sur ce graphe. Nous proposons de définir des schémas superposables à cette structure, que nous nommons schémas de transduction compositionnels (STC). Chaque schéma peut être activé par un ensemble de filets sémantiques, pour enrichir la structure en affinant son interprétation.

La définition des STC permet ainsi le paramétrage d'un algorithme d'analyse dont la visée est l'enrichissement des filets sémantiques. Intuitivement, chacun de ces filets produit une interprétation partielle du graphe, que les schémas complètent. *In fine*, une

interprétation finale est éventuellement produite, interprétation associée au type de sortie attendu (type output).

Dans la suite, les notions de requête logique et d'opérateur de transduction sont proposées et associées pour définir les schémas de transduction (STC).

4.2.1 Requête logique

L'activation d'un STC conduit à l'application d'une certaine opération, dite de transduction, sur un ou plusieurs filets sémantiques. Cette opération est définie un peu plus loin. L'exécution d'une transduction est conditionnelle. Ainsi, chaque STC peut être actif, ou au contraire, inactif, en fonction de circonstances à spécifier.

Nous proposons de délimiter les critères d'activation des STC sous la forme de formules logiques, que nous nommons requêtes logiques. Précisément, une requête logique est une formule logique conjonctive du premier ordre respectant la définition inductive 5.

Définition 5 *Requête logique.*
Une requête logique est une formule sur les variables libres $x_1, \ldots, x_n$ définie inductivement par la grammaire suivante en BNF :

$$\mathcal{R}[x_1, \ldots, x_n] ::= p(x_i) | p(x_i, x_j) | \mathcal{R} \wedge \mathcal{R}.$$

Une requête logique est un élément $req \in \mathcal{R}$.

Exemple 19 *Les exemples suivants permettent de capturer des filets sémantiques dont il est possible de donner une interprétation intuitive :*

- $req_1[x] = entity(x)$ *: filet associé au type entity ;*
- $req_2[x, y] = modality(x) \wedge property(y) \wedge arg(x, y)$ *: filets exprimant une relation entre une modalité et une propriété ;*
- $req_3[x, y, z] = modality(x) \wedge property(y) \wedge context(z) \wedge arg(x, y) \wedge condition(x, z)$ *: filets associant une modalité, une propriété et un contexte.*

Pour définir la relation de satisfaction, nous considérons la classe de tous les graphes enrichis de filets sémantiques $\mathcal{C}_{\mathcal{GF}}$, et le langage de requêtes $\mathcal{R}$. Cette définition est basée sur la notion de propriété appliquée aux filets (voir section 4.1.2).

Définition 6 *Relation de satisfaction.*
Soit $\mathcal{T}$ un ensemble de types. Soit $(G, (\mathfrak{f}_1, \ldots, \mathfrak{f}_n)) \in \mathcal{C}_{\mathcal{GF}}$, avec $G = (S, A, \kappa, \eta)$ et $\forall i \in \{1, \ldots, n\}, \mathfrak{f}_i = (X_i, x_{0,i}, t_i, v_i)$. Soit p une propriété associée à un ensemble de types $C_p \subseteq \mathcal{T}$.
La relation de satisfaction $\models_{\mathcal{GFR}} \subseteq \mathcal{C}_{\mathcal{GF}} \times \mathcal{R}$ est définie inductivement par :

- $(G, (\mathfrak{f}_1, \ldots, \mathfrak{f}_n)) \models p(x_i)[x_1, \ldots, x_n]$ *si et seulement si* $t_i \in C_p$;
- $(G, (\mathfrak{f}_1, \ldots, \mathfrak{f}_n)) \models p(x_i, x_j)[x_1, \ldots, x_n]$ *si et seulement si* $(x_{0,i} \rightarrow^p x_{0,j})$;
- $(G, (\mathfrak{f}_1, \ldots, \mathfrak{f}_n)) \models r_1[x_1, \ldots, x_n] \wedge r_2[x_1, \ldots, x_n]$
  si et seulement si $(G, (\mathfrak{f}_1, \ldots, \mathfrak{f}_n)) \models q_1[x_1, \ldots, x_n]$ *et* $(G, (\mathfrak{f}_1, \ldots, \mathfrak{f}_n)) \models q_2[x_1, \ldots, x_n]$.

Remarque. La notation $\mathfrak{f} \models r$ peut être utilisée s'il n'y a pas d'ambiguïté sur le graphe. Notons également que lorsque la propriété p ne correspond qu'à un seul type, c'est à dire lorsque $C_p = \{p\}$, alors la requête $p(x)$ est satisfaite par un filet $\mathfrak{f}$ si le type de ce filet est égal à cette propriété.

Exemple 20 *La figure 4.6 présente un graphe sémantique, dénoté $\mathcal{G}_1$, pour l'énoncé "The entry gate may open if a ticket has been issued.". Chaque sommet peut être associé à un filet élémentaire (ces filets sont notés $\mathfrak{f}_1, \mathfrak{f}_2, \ldots$).*

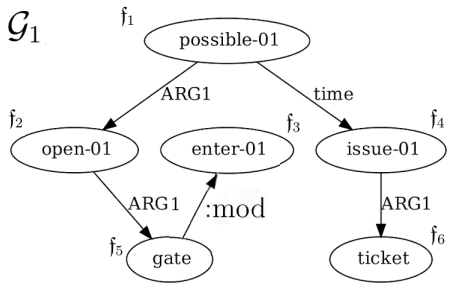


FIGURE 4.6 – Graphe pour l'énoncé "*The entry gate may open if a ticket has been issued.*"

Considérons la propriété "entity", propriété associée à l'ensemble des concepts se rapportant à une entité quelconque ($C_{entity} = \{gate, ticket, \ldots\}$). La requête $r[x] = entity(x)$ est satisfaite par les filets $\mathfrak{f}_5$ et $\mathfrak{f}_6$ (figure 4.7), respectivement associés aux concepts "gate" et "ticket", présents dans l'ensemble C_{entity}.

Considérons la propriété modality, qu'il est possible d'associer à l'ensemble des modalités $C_{modality} = \{possible, obligatory, necessary, \ldots\}$. La requête $r'[x, y] = modality(x) \wedge times(x, y)$ est satisfaite par les filets $(\mathfrak{f}_1, \mathfrak{f}_4)$ (figure 4.8). Le type de $\mathfrak{f}_1$ appartient à l'ensemble $C_{modality}$. Les racines de $\mathfrak{f}_1$ et $\mathfrak{f}_4$ sont reliées par la relation "times".

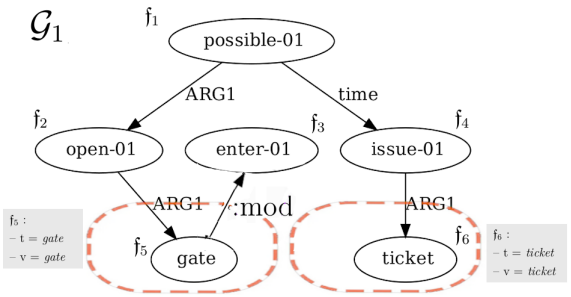


FIGURE 4.7 – $\mathfrak{f}_5 \models entity(x)$ et $\mathfrak{f}_6 \models entity(x)$

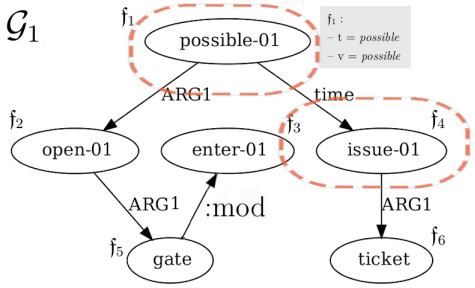


FIGURE 4.8 – $(\mathfrak{f}_1, \mathfrak{f}_4) \models modality(x) \wedge times(x, y)$

4.2.2 Opérateur de transduction

Les filets de transduction permettent de construire des ensembles reliés aux sommets d'un graphe. Ces ensembles sont associés à des attributs, un type et une valeur, qui donnent une interprétation partielle du graphe. Pour affiner ces interprétations partielles, nous proposons une opération pour composer les filets sémantiques. Cette opération prend la forme d'opérateurs de transduction, dont la définition est basée sur deux fonctions : une fonction de typage et une fonction de valuation.

Définition 7 *Fonctions de typage τ et de valuation ν.*

Une fonction de typage $\tau : t_1, \ldots, t_n \rightarrow t$ est une application monotone qui prend en paramètre une liste de types, et qui retourne un nouveau type.

Une fonction de valuation $\nu : t_1, \ldots, t_n, v_1, \ldots, v_n \rightarrow v$ est une application monotone qui prend en paramètre une liste de types et de valeurs, et qui retourne une nouvelle valeur.

Un opérateur de transduction, noté tr, est une opération qui retourne un nouveau filet. L'ensemble des sommets capturés est construit par union, tandis que les attributs (type,

valeur) du nouveau filet sont obtenus en fonction des attributs (type, valeur) des filets composés. Chaque opérateur de transduction est ainsi associé à deux fonctions permettant de calculer respectivement le nouveau type et la nouvelle valeur du filet sémantique sortant.

Définition 8 *Opérateur de transduction $tr_{\tau,\nu}$.*

Soit $\tau : t_1, \ldots, t_n \rightarrow t$ une fonction de typage, et $\nu : t_1, \ldots, t_n, v_1, \ldots, v_n \rightarrow v$ une fonction de valuation. Soit $\mathfrak{f}_1 = (X_1, x_{0,1}, t_1, v_1)$, …, $\mathfrak{f}_n = (X_n, x_{0,n}, t_n, v_n)$ des filets sémantiques.
La transduction de $\mathfrak{f}_1 = (X_1, x_{0,1}, t_1, v_1)$, …, $\mathfrak{f}_n = (X_n, x_{0,n}, t_n, v_n)$ suivant l'opérateur $tr_{\tau,\nu}$, notée $tr_{\tau,\nu}(\mathfrak{f}_1, \ldots, \mathfrak{f}_n)$, produit un filet sémantique $\mathfrak{f} = (X, x_0, t, v)$ tel que :

- $X = \bigcup X_1, \ldots, X_n$;
- $x_0 = x_{0,1}$;
- $t = \tau(t_1, \ldots, t_n)$, *avec, pour tout* $t_i \in (t_1, \ldots, t_n)$, $t \geq t_i$;
- $v = \nu(t_1, \ldots, t_n, v_1, \ldots, v_n)$.

Intuitivement, les fonctions de typage et de valuation donnent une certaine signification au nouveau filet produit. Ainsi, l'exemple 21 montre la formation d'un filet capturant un sommet, de type *modality*, et dont la valeur est une définition normalisée de la modalité correspondant au sommet capturé. L'opérateur de l'exemple 22 permet d'obtenir un nouveau filet par composition de plusieurs filets. Ce nouveau filet caractérise une propriété (type *property*), dont la valeur désigne le nom de la propriété. Dans cet exemple, la fonction de valuation est une fonction de nommage.

De fait, en pratique, le nouveau type ne dépend pas nécessairement des types passés en paramètres. De même, la nouvelle valeur est généralement obtenue par composition des valeurs passées en paramètres, en fonction des types. La fonction de valuation peut être vue, dans certains cas, comme une fonction de nommage. D'une manière générale, la valeur obtenue définit un nouvel énoncé, correspondant au type attendu, qui deviendra, in fine, l'énoncé complet attendu.

Propriété 1 *Les opérateurs de transduction définissent des applications monotones.*

Preuve 1 *La preuve découle naturellement du caractère monotone de la fonction de typage, de l'union des ensembles et de la fonction de valuation.*

Exemple 21 *Un typage permet de caractériser une sous-partie du graphe correspondant, par exemple, à une modalité, une propriété, ou une entité. La fonction de valuation définit l'énoncé correspondant à ce type. L'opérateur* $tr_{modality,\nu_{mod}}$ *peut être défini pour obtenir un filet sémantique de type modality et de valeur* $\nu_{mod}(x)$*. La fonction* $\nu_{mod}(x)$ *définit une valeur normalisée d'une modalité en fonction du concept* x *:* $\nu_{mod}(x) = POSSIBLE$ *si* $x \in \mathcal{C}_{possible}$*,* $NECESSARY$ *si* $x \in \mathcal{C}_{necessary}$*,* $IMPOSSIBLE$ *si* $x \in \mathcal{C}_{impossible}$*, où* $\mathcal{C}_{possible}, \mathcal{C}_{necessary}, \mathcal{C}_{impossible}$ *sont des ensembles de concepts. La figure 4.9 montre le résultat de l'application de cet opérateur sur le filet* $\mathfrak{f}_1$ *du graphe* $\mathcal{G}_1$*.*

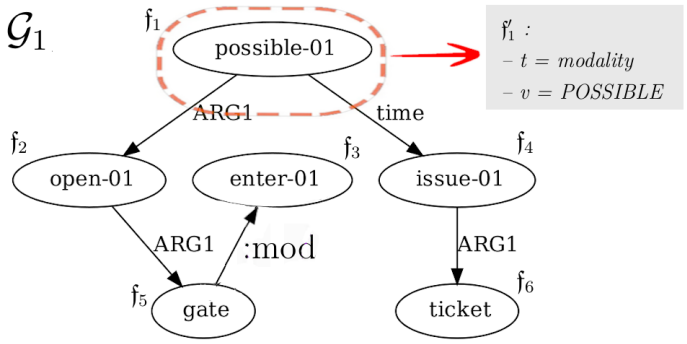


FIGURE 4.9 – Le filet $\mathfrak{f}'_1$ détermine le type et la valeur normalisée du filet élémentaire $\mathfrak{f}_1$.

Exemple 22 *L'opérateur* $tr_{property,\nu_{prop}}$ *permet de construire un filet sémantique de type property et de valeur* $\nu_{prop}(x, y)$*. La fonction* $\nu_{prop}(x, y)$ *définit une valeur normalisée de la propriété en fonction des valeurs* x *et* y *:* $\nu_{prop}(x, y) = x - y$*. Cette valeur désigne le nom de la propriété (la fonction de valuation est une fonction de nommage). L'application de cet opérateur sur les filets* $\mathfrak{f}_4$ *et* $\mathfrak{f}_6$ *du graphe* $\mathcal{G}_1$ *est illustrée par la figure 4.11.*

4.2.3 Schéma de transduction compositionnel (STC)

Il est finalement possible de définir des schémas en associant requête logique et opérateur de transduction. Un *STC* définit une opération qui s'applique sur une liste de filets ciblés par la requête logique, à la condition que celle-ci soit satisfaite. Le résultat est un nouveau filet.

Définition 9 *Schéma de transduction compositionnel (STC).*

Un schéma de transduction compositionnel, noté STC, est une paire $STC = (req, tr)$ *telle que :*

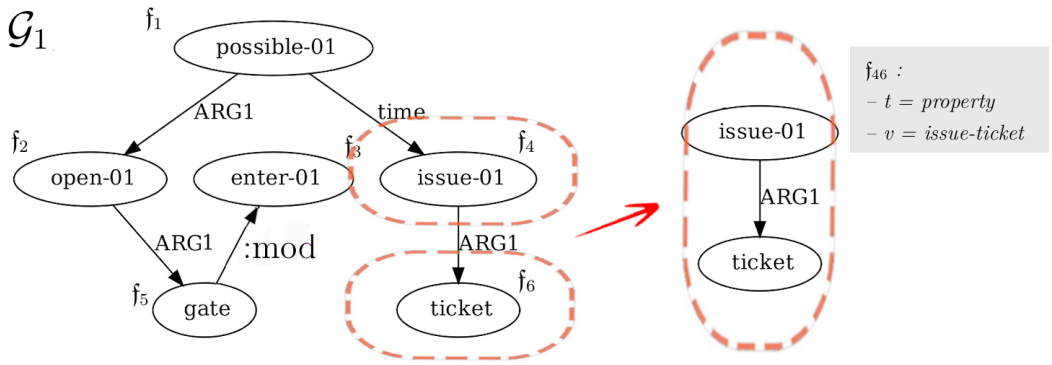


FIGURE 4.10 – Le filet $\mathfrak{f}_{46}$ est obtenu par composition des filets $\mathfrak{f}_4$ et $\mathfrak{f}_6$. L'opération appliquée permet d'obtenir un nouvel ensemble de sommets, associé à un nouveau type et une nouvelle valeur, en fonction des types et valeurs de $\mathfrak{f}_4$ et $\mathfrak{f}_6$.

— *req est une requête logique ;*

— *tr est un opérateur de transduction.*

Exemple 23 *Le schéma* $s_1 = (req_1, tr_1)$*, avec* $req_1 = modality(x)$ *et* $tr_1 = (modality, \nu_{mod})$*, s'applique sur tout filet dont le concept correspond à une modalité. Le nouveau filet obtenu est associé au type modality et à la valeur donnée par l'application de la fonction* ν_{mod}*, comme sur la figure 4.9 (section précédente). La figure 4.11 montre l'application du schéma* $s_2 = (req_2, tr_2)$*, avec* $req_2 = modality(x) \wedge property(y) \wedge context(z) \wedge arg(x, y) \wedge condition(x, z)$ *et* $tr_2 = (mpc, \nu_{mpc})$*. La valeur associée au filet est obtenue par application de la fonction* ν_{mpc}*, produisant un énoncé au format attendu.*

$\mathfrak{f}_{123456}$:
– t = *mpc*
– v = *(POSSIBLE, open-enter-gate, issue-ticket)*

Sémantique. Les *STC* produisent des ensembles de filets sémantiques en partant d'un ensemble initial non nul. L'application récursive de *STC* en partant d'un ensemble quelconque permet d'enrichir cet ensemble. Comme vu avec la définition 4, le poids d'un ensemble de filets sémantiques donne une mesure quantitative du volume d'interprétations contenu. Le poids maximal, défini par le produit du cardinal des types et du cardinal des sommets, donne un critère de terminaison pour l'application récursive. De plus, les *STC* peuvent être considérés monotones par extension de cette propriété aux opérateurs de transduction.

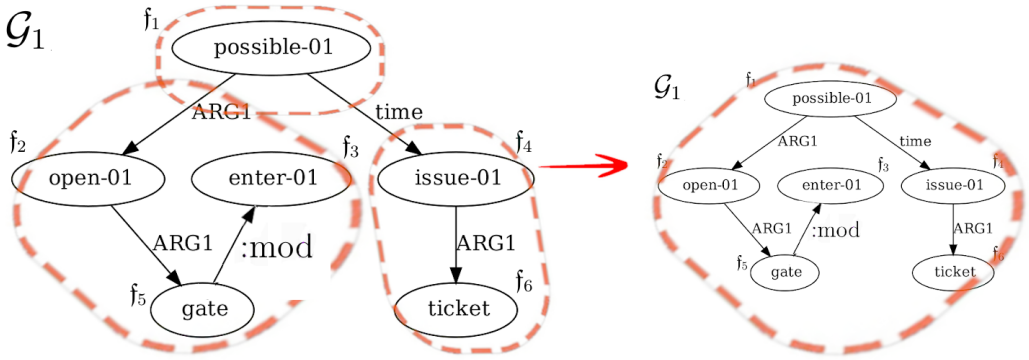


FIGURE 4.11 – Application du schéma $s_2 = (req_2, tr_2)$

4.3 Algorithme d'analyse sémantique

L'objectif suivi est la mise en œuvre d'un processus d'analyse de documents textuels, composés d'énoncés en langage naturel, et intégrant une capacité d'interprétation sur les données traitées. Les représentations sémantiques, telles que présentées dans le chapitre 3, sont exploitées pour obtenir une première représentation symbolique de ces énoncés. À partir de là, l'analyse sémantique vise l'interprétation de ces représentations. Il s'agit d'élaborer un cadre de calcul permettant de transformer la structure obtenue en prenant en compte sa signification, et donc celle des énoncés traités. Celle-ci se reflète notamment dans les concepts et les relations qui peuvent être extraits du texte étudié.

L'algorithme 1 définit une procédure d'analyse par transduction sémantique. Il s'applique sur toute structure sémantique respectant les définitions de la section 4.1, qu'il est possible d'associer à un ensemble initial de filets sémantiques élémentaires. La mise en œuvre de l'algorithme permet d'enrichir cet ensemble avec de nouveaux filets, obtenus par composition des filets déjà définis. Les paramètres sémantiques avancés dans la section 4.2 sont exploités dans ce but. La requête précise les conditions d'application du calcul, tandis que l'opérateur produit les nouveaux filets sémantiques.

Le processus d'analyse vise la construction d'un ensemble de filets sémantiques autorisant une interprétation de plus en plus riche. La notion de poids introduit pour les ensembles de filets sémantiques est utilisée pour définir la terminaison du calcul.

Exemple 24 *Les premières règles permettent d'ajuster les types et normaliser les valeurs pour chaque filet élémentaire, ce que l'on peut observer, sur la figure 4.12, pour les filets* $\mathfrak{f}_1$, $\mathfrak{f}_4$ *et* $\mathfrak{f}_6$.

Le traitement est ensuite itératif. Les nouveaux filets s'obtiennent par composition. La

Algorithm 1 ANALYSE PAR TRANSDUCTION SÉMANTIQUE

Entrée : $(\mathcal{G}, \mathcal{F})$, $\mathcal{E}_{STC}$ ▷ graphe, filets de base et schémas de transduction
Sortie : $(\mathcal{G}, \mathcal{F}^+)$ ▷ enrichissement des filets sémantiques

$poids = 0$
$\mathcal{F}^+ \leftarrow \mathcal{F}$ ▷ initialisation

while $||\mathcal{F}^+|| > poids$ **do**
  $poids = ||\mathcal{F}^+||$
  $\mathcal{F} \leftarrow \mathcal{F}^+$
  for all $s = (req, tr) \in \mathcal{E}_{STC}$ **do** ▷ pour tout schéma de transduction
    for all $F \subseteq \mathcal{F}$ **do** ▷ pour tout ensemble de filets sémantiques
      if $F \models req$ **then**
        $\mathcal{F}^+ = \mathcal{F}^+ \cup tr(F)$ ▷ ajout d'un nouveau filet sémantique
      end if
    end for
  end for
end while

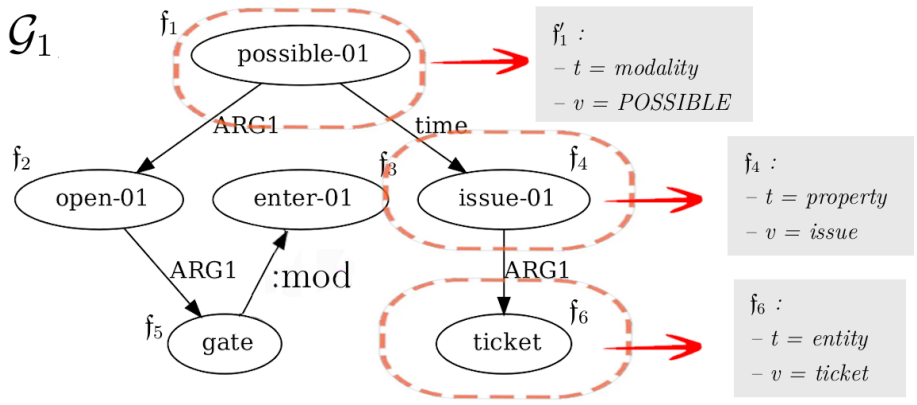


FIGURE 4.12 – Ajustement des types et des valeurs pour les filets $\mathfrak{f}_1$, $\mathfrak{f}_4$ et $\mathfrak{f}_6$.

figure 4.14 montre ainsi le calcul des filets $\mathfrak{f}_{235}$ *et* $\mathfrak{f}_{46}$*, respectivement à partir des filets* $\mathfrak{f}_2$*,* $\mathfrak{f}_3$*,* $\mathfrak{f}_5$ *et* $\mathfrak{f}_4$*,* $\mathfrak{f}_6$*.*

Finalement, la figure 4.14 donne le résultat final, obtenu par application du schéma du schéma $s_2 = (req_2, tr_2)$ *(exemple 23) sur les filets* $\mathfrak{f}'_1$*,* $\mathfrak{f}_{235}$ *et* $\mathfrak{f}_{46}$*.*

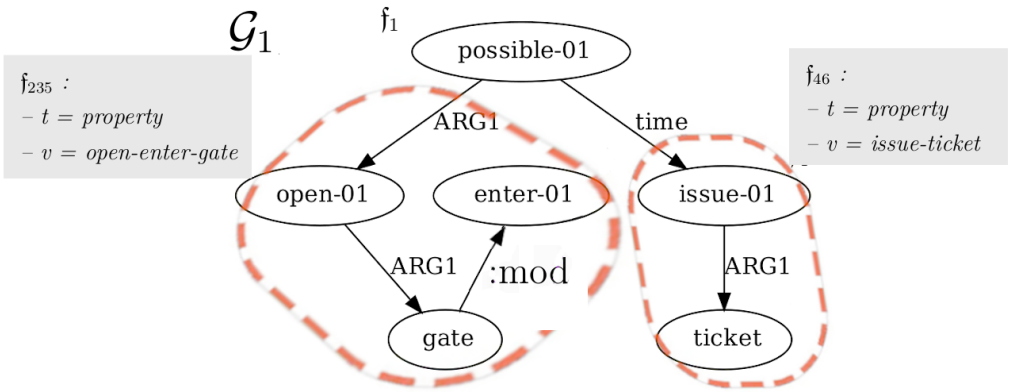


FIGURE 4.13 – Les filets $\mathfrak{f}_{235}$ et $\mathfrak{f}_{46}$ sont obtenus par composition, respectivement à partir des filets $\mathfrak{f}_2$, $\mathfrak{f}_3$, $\mathfrak{f}_5$ et $\mathfrak{f}_4$, $\mathfrak{f}_6$.

$\mathfrak{f}_{123456}$:
– t = *mpc*
– v = *(POSSIBLE, open-enter-gate, issue-ticket)*

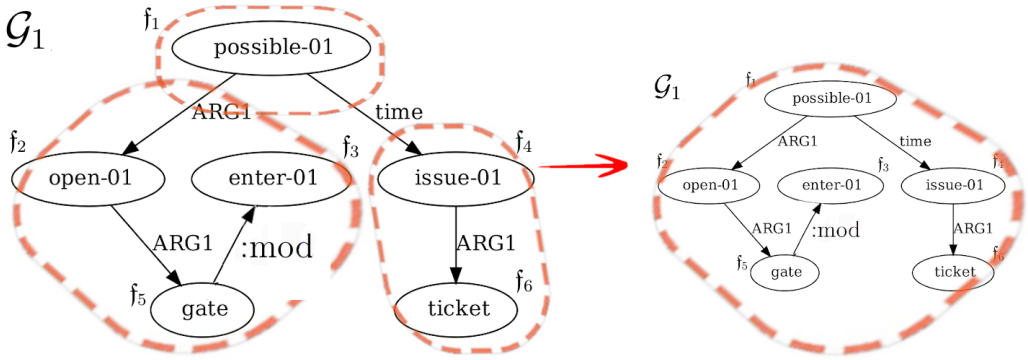


FIGURE 4.14 – Application du schéma $s_2 = (req_2, tr_2)$

Terminaison Les opérations de transduction sont appliquées tant qu'elles permettent d'enrichir l'interprétation du graphe sémantique, en introduisant des filets d'un nouveau type, ou en raffinant l'interprétation pour un type donné. Le poids de l'ensemble des filets évolue en fonction du nombre de types ou du nombre de sommets couverts pour chaque type. Il est donc borné par $p_{max} = Card(T) \times Card(S)$. La terminaison de l'algorithme est assurée en considérant le caractère monotone des opérateurs de transduction. La formalisation attendue est acquise si l'interprétation finale obtenue est associée au type terminal, c'est à dire au type de la formalisation attendue. Si ce n'est pas le cas, l'énoncé est rejeté.

Correction. L'algorithme génère un ensemble de filets sémantiques, que le typage et la relation d'ordre permettent de classer. La correction dépend des règles de transduction et du résultat attendu. La vérification est expérimentale (voir section 4.4). Quelques ajustements techniques sont également nécessaires pour une bonne prise en compte de la relation d'ordre.

Couverture des phénomènes linguistiques. La couverture des phénomènes linguistiques découlent de la représentation pivot utilisée et de la définition des schémas de transduction. Il n'y a pas de limitation particulière liée aux schémas de transduction, qu'il est possible d'étendre, en ajoutant de nouveaux schémas, pour couvrir l'ensemble des phénomènes représentés dans le graphe traité. Nos expérimentations (section 4.4) ont ainsi permis de montrer le traitement efficace d'un sous-langage significatif de la langue anglaise.

Exemple 25 *Le traitement de l'énoncé "It is not possible to ask for a drink in standby mode." est proposé pour exemple. La figure 4.15 montre l'analyse initiale d'un graphe sémantique correspondant à l'énoncé. Chaque nœud est associé à un type déterminé en fonction de son concept (attaché dans le graphe initial) et des définitions suivantes :*

- $\mathcal{T} = \{m1, m2, p1, p2, e1, e2, c, mpc\}$,
- $\mathcal{C}_{m1} = \{possible, \ldots\}$,
- $\mathcal{C}_{p1} = \{request, drink, standby, \ldots\}$,
- $\mathcal{C}_{e1} = \{mode, \ldots\}$.

Plusieurs schémas de transduction peuvent alors s'appliquer pour faire apparaître les filets de la figure 4.16. Les valeurs sont calculées à l'aide des fonctions adéquates : une fonction $negMod(x)$ pour obtenir une modalité négative, une fonction $prop(x, y)$ pour obtenir le nom de la propriété construite à partir de deux filets. Le filet $\mathfrak{f}_{11}$, de type $m2$, spécifie ainsi la modalité après prise en compte de la négation. Ce filet est obtenu à partir du filet $m1$, capturée par la requête $modality(x) \wedge negPolarity(x)$, et par application d'un opérateur utilisant la fonction $negMod(x)$.

Finalement, il est possible d'obtenir un filet de type mpc en appliquant le schéma $s_{mpc1} = (r, t)$ tel que :

- $r[x, y, z] = modality(x) \wedge property(y) \wedge context(z)$;
- $t = (mpc, \nu_{mpc})$, *avec* $\nu_{mpc}(x, y, z) = (x, y, z)$.

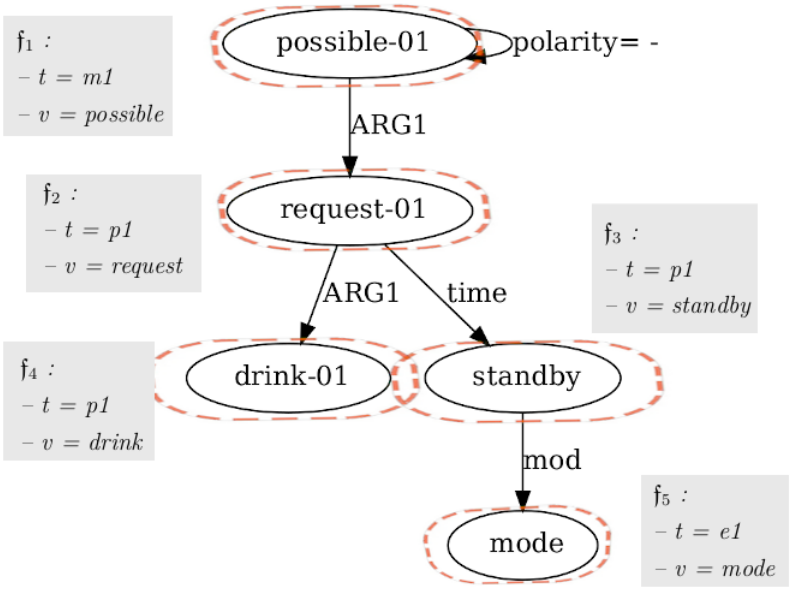


FIGURE 4.15 – Traitement de l'énoncé "It is not possible to ask for a drink in stand-by mode." (initialisation)

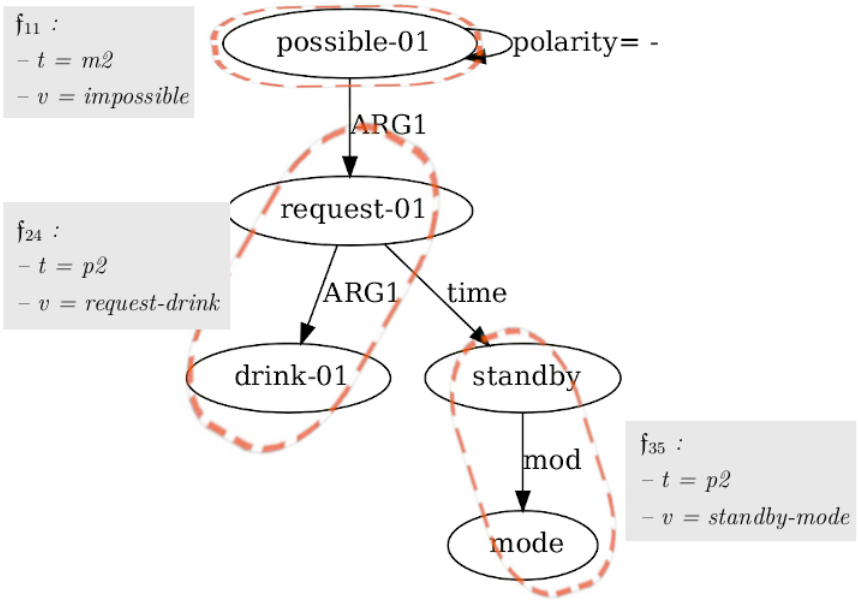


FIGURE 4.16 – Traitement de l'énoncé "It is not possible to ask for a drink in stand-by mode." (étape intermédiaire)

Le résultat est illustré par la figure 4.16. La formalisation au format MPC correspond à la valeur du filet final (de type mpc) : $(impossible, request - drink, standy - mode)$. *L'algorithme permet de produire un ensemble de filets. L'interprétation de l'énoncé comme*

exigence exprimable dans le format MPC est vérifiée par la présence d'un filet de type mpc. La relation d'ordre sur les filets, en partie basée sur la relation d'ordre sur les types, permet de retenir les filets les plus précis (donc les plus pertinents). Le filet de type m2, qui spécifie la modalité après prise en compte de la négation, est ainsi favorisé par rapport au filet de type m1 (modalité avant prise en compte de la négation). Il en est de même pour les filets de type p2 par rapport aux filets de type p1.

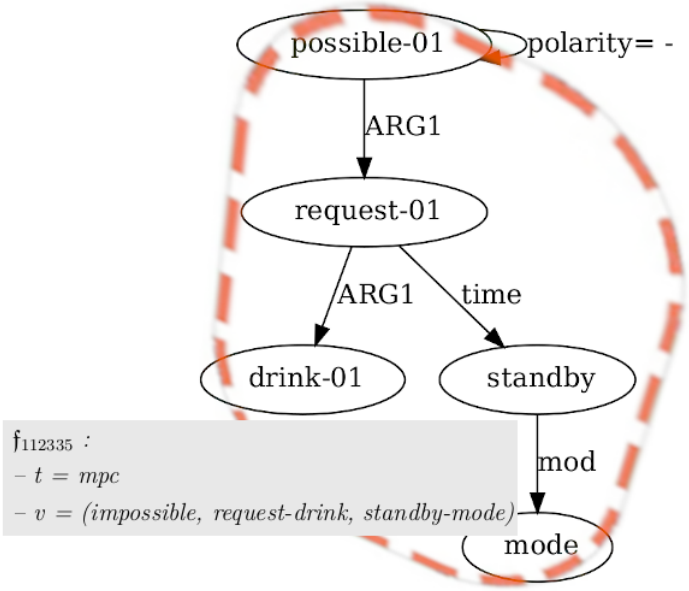


FIGURE 4.17 – Traitement de l'énoncé "It is not possible to ask for a drink in stand-by mode." (étape finale)

4.4 Expérimentation

Nos expériences se sont concentrées sur les tâches d'extraction et de formalisation des exigences. Le problème est l'analyse d'une suite de phrases. Certaines phrases définissent une exigence attendue, c'est-à-dire une règle à extraire, et d'autres non. Dans ce contexte, l'objectif est de proposer un processus d'analyse permettant de classer les énoncés, et d'en extraire les exigences pertinentes pour une méthode formelle spécifique. Cette méthode s'accompagne d'une procédure pour formaliser ces exigences en utilisant un format précis. Nos expérimentations ont été soutenues par un format *ad hoc* permettant de définir une relation formelle entre un contexte et une propriété associée à une modalité. Ce format est dénoté MPC (modalité, propriété, contexte). Un élément du langage MPC est un triplet

<*modalité, propriété, contexte*> tel que :

— la *modalité* spécifie la modalité déontique associée à la propriété, en termes d'obligation (*necessary*), d'interdit (*interdiction*) et de possibilité (*possible*) ;

— la *propriété* est une proposition caractérisant un événement ou une propriété sur laquelle s'applique la règle ;

— le *contexte* est une formule de logique propositionnelle spécifiant les conditions d'application de la règle.

4.4.1 Implémentation

La méthodologie proposée a été entièrement implémentée et évaluée à l'aide des langages Python et Java. Le module principal, écrit en Python, permet de contrôler diverses opérations. Les opérations de prétraitement sont conventionnelles. Elles comprennent le processus de découpage et de segmentation des phrases en unités syntaxiques (tokenisation) et en étiquettes grammaticales (POS tagging). Toutes ces opérations sont effectuées avec l'outil Stanford CoreNLP [48]. L'analyse syntaxico-sémantique pour la génération de structures AMR peut être utilisée en choisissant l'un des deux algorithmes proposés. L'implémentation de ces algorithmes est une adaptation des outils CAMR [82] et STOG [49], dont le code est sous licence libre.

Processus de transduction. Le module ATP (Abstract Transduction Process) est la principale contribution de notre mise en œuvre. Ce module a été développé en java. Il intègre les étapes de transduction sémantique. La configuration de l'outil comprend la définition d'une ontologie (module OntoMap) et la définition des schémas de transduction. Un fragment du corpus PropBank [50] est intégré dans l'ontologie. Il est suffisant pour couvrir différents phénomènes linguistiques. Précisément, les entités et les concepts marquant une propriété sont directement reconnus. D'autres concepts sont associés aux phénomènes linguistiques que nous avons cherché à traiter en particulier : les modalités temporelles (*after*, *before*), les modalités déontiques (*possible*, *necessary*, *interdiction*) et les connecteurs logiques (*and*, *or*). Ces différents groupes de concepts alimentent des ensembles qui permettent de typer les filets sémantiques élémentaires. Les schémas de transduction sont utilisés lors de l'étape d'analyse par transduction sémantique, pour aboutir aux formalisations attendues.

Alphabets Les propriétés du système à modéliser peuvent être définies sous la forme de propositions atomiques. Les exigences comportementales caractérisent les évolutions possibles de l'état d'un système, et la survenue d'un événement (action du système ou événement extérieur). Ces éléments sont fixés par les propriétés d'état et événementielles, et regroupés dans un alphabet. Quelques mots-clés peuvent également être inclus, par exemple pour les modalités déontiques (possible, nécessaire), les modalités temporelles (après, avant) ou les connexions logiques (et, ou). Tous ces éléments peuvent être directement prélevés dans la représentation pivot en utilisant une ontologie caractérisant les concepts de la représentation, et un ensemble de méthodes regroupées dans une classe spécifique (classificateur). Un traitement complémentaire pourrait aider à organiser l'alphabet pour construire une taxonomie, en regroupant, par exemple, les modalités (nécessaires, possibles), les entités (porte ouverte, ticket) ou les propriétés (porte ouverte, ticket d'émission).

4.4.2 Paramétrage

Le processus de transduction sémantique combine deux tâches : l'extraction d'informations et la formalisation des informations extraites. Les critères d'extraction sont définis dans le schéma de transduction au travers des requêtes logiques. La formalisation attendue est définie au même niveau, à l'aide des opérateurs de transduction. Ces opérateurs sont spécifiés directement dans les fichiers techniques prévus. Il est possible de définir plusieurs ensembles de paramètres et différents critères d'extraction.

Typage. Les filets de base sont générés avec le typage initial en paramétrant le module classificateur. Un ensemble de types est fixé, chaque type pouvant être associé à un ensemble de concepts ou valeurs. Le système de typage suivant peut servir comme base de départ :

- $\mathcal{T} = \{m1, m2, temp, lc, adv, p1, p2, p3, e1, e2, c1, c2, mpc, \ldots\}$,
- $\mathcal{C}_{m1} = \{possible, necessary, allow, prohibit, \ldots\}$,
- $\mathcal{C}_{temp} = \{after, before, \ldots\}$,
- $\mathcal{C}_{lc} = \{and, or, \ldots\}$.

Les propositions verbales et les entités peuvent être différenciées par heuristique dans un graphe AMR, l'alternative étant de mettre à jour le système de typage et les ensembles associés. Ces paramètres peuvent être ajustés au fur et à mesure des expérimentations

pour affiner les résultats. Des sous-ensembles sont également exploités pour faciliter la définition des schémas de transduction.

Requêtes. La construction des schémas de transduction passe par la définition des requêtes logiques et des opérateurs de transduction. Les requêtes sont fondées sur le système de types pour capturer des ensembles de filets suivant différentes interprétations, permettant de faire émerger de nouveaux filets correspondant aux éléments attendus. Dans notre cas, il s'agit des modalités déontiques, des propriétés d'états et des événements (sous la forme de propositions atomiques), suivis des relations entre ces éléments (contexte et *mpc*). Le tableau 4.1 regroupe quelques requêtes utiles. Des ajustements techniques permettent de simplifier les formules, comme pour la requête r_6 où la relation $context(y, z)$ est satisfaite s'il existe un arc reliant les filets y et z et étiqueté par *time* ou *condition*.

Réf.	*Requête*	*Commentaire*
r_1	$mod(x)$	filet correspondant à une modalité
r_2	$p1(x)$	filet correspondant à une propriété simple (proposition verbale)
r_3	$e1(x)$	filet correspondant à une entité simple
r_4	$p1(x) \wedge e1(y) \wedge arg1(x, y)$	filets correspondant à une relation entre une propriété et une entité
r_5	$p(x) \wedge p(y) \wedge time(x, y)$	filets correspondant à une relation entre deux propriétés, la seconde propriété définissant un contexte
r_6	$m(x) \wedge p(y) \wedge c(z) \wedge context(y, z)$	filets correspondant à une relation *MPC*
r_i	...	...

TABLE 4.1 – Quelques requêtes utilisées dans les schémas de transduction

Opérateurs de transduction. Les opérateurs de transduction permettent de faire émerger les nouveaux filets sémantiques par composition de plusieurs filets. Ces opérateurs peuvent également être unaires pour associer un nouveau type et une nouvelle valeur à un filet (ainsi dupliqué). Quelques exemples d'opérateurs sont donnés dans le tableau 4.2.

Schémas de transduction (*STC*). Les schémas de transduction sont finalement définis en associant les formules logiques et les opérateurs. Les éléments des tableaux 4.1 et 4.2 peuvent ainsi être associés pour spécifier les schémas $c_1 = (r_1, p_1)$, $c_2 = (r_2, p_2)$,

Réf.	Typage τ	Valuation ν	*Commentaire*
t_1	$m2$	$\nu_m(x) = modNorm(x)$	nommage d'une modalité
t_2	$p1$	$\nu_p(x) = propNorm(x)$	nommage d'une propriété
t_3	$e1$	$\nu_e(x) = entNorm(x)$	nommage d'une entité
t_4	$p2$	$\nu_p(x, y) = propNorm(x, y)$ à partir de deux propriétés	nommage d'une propriété
t_5	c	$\nu_c(x) = x$	définition d'un contexte
t_6	mpc	$\nu_{mpc}(x, y, z) = (x, y, z)$	définition d'une relation MPC
t_i	...	...	...

TABLE 4.2 – Quelques opérateurs utilisés dans les schémas de transduction

etc. D'autres requêtes, opérateurs et schémas complètent les éléments donnés en exemple pour obtenir un système efficace.

Les schémas utilisés dans nos expérimentations ont ainsi permis de traiter un sous-ensemble significatif de la langue anglaise, intégrant des phrases avec des modalités simples, des relations logiques et des éléments de temporalités. Les négations sont également prises en considération. Il n'y a pas de contraintes particulières sur les propriétés et les entités. Toute proposition verbale est considérée comme marqueur de propriétés. Les concepts qui ne correspondent à aucun phénomène linguistique et à aucune proposition verbale sont considérés comme marqueurs d'entités. Ce sous-ensemble n'a pas de limites fixes : il peut être facilement étendu par mise à jour des paramètres du système de transduction.

4.4.3 Évaluation

Les corpus d'évaluation ont été constitués à partir de diverses études de cas. Ils couvrent différents domaines d'application, en s'inspirant de phrases que l'on peut trouver dans des spécifications réelles. Ces phrases sont reprises dans les exemples donnés dans ce manuscrit. Ces études de cas forment différents corpus, pour un total de 150 phrases. Ils couvrent plusieurs phénomènes linguistiques tels que les conditions, les modalités, la négation, la temporalité et les connecteurs logiques.

Les résultats obtenus sont rassemblés dans le tableau 4.3. La taille des corpus est précisée. La colonne *AMR Score* donne le résultat observé après la première phase de traitement (AMR Parsing), tandis que la colonne *MPC Score* correspond au résultat final, évalué après la deuxième phase (Semantic Transduction). La lecture du tableau permet d'évaluer la performance du processus dans les différentes tâches attendues : la

classification avec le score de rappel, et la formalisation avec le score de précision.

Corpus	*Size*	*AMR Score*	MPC Precision	MPC Recall	*MPC Score*
Parking Access	30	0.73	0.75	0.74	0.75
Vending Machine	20	0.88	0.84	0.70	0.82
Russian Dolls	30	0.83	0.88	0.75	0.81
Autonom. Vehicle	20	0.73	0.74	0.72	0.73
About Weather	50	0.80	0.78	0.69	0.75
TOTAL (average)	150	0.79	0.79	0.72	**O.77**

TABLE 4.3 – Résultats

Les résultats obtenus montrent une grande efficacité du processus de transduction, tant en termes de rappel que de précision. Si l'analyse AMR produit un graphe conforme, le résultat final, atteint après transduction sémantique, aboutit également à une production correcte (à quelques exceptions mineures près). Cette expérimentation conduit à la conclusion que les principes mis en œuvre fonctionnent sur des phrases simples, en limitant les phénomènes linguistiques. Notons que l'évaluation du *F-score* est sévère pour un processus sémantique. Elle ne permet pas de traduire l'écart entre la formalisation attendue et la formalisation obtenue, lorsque cette dernière est inexacte. D'autres critères, tels que le score *Smatch*, pourraient également être utilisés pour affiner cette évaluation. Mais cela ne semble pas être absolument nécessaire dans notre cas. La comparaison des représentations obtenues après la première phase (AMR) et la deuxième phase (MPC) montre assez clairement l'efficacité du processus.

Les résultats montrent aussi une très bonne couverture des phénomènes linguistiques (modalité, négation, temporalité, etc). Celle-ci peut être augmentée en ajustant le paramétrage du système de transduction. De même, le traitement de phrases plus complexes est envisageable par extension.

Nous avons ainsi expérimenté l'analyse d'énoncés axée sur les modalités déontiques, en les traduisant dans un format *ad hoc* (MPC). Il est tout à fait possible de traiter d'autres types de déclarations. L'adaptation de l'outil porterait essentiellement sur la définition de nouveaux schémas de transduction sémantique (STC).

Ces résultats établissent une première validation des principes proposés dans ce chapitre. La mise en œuvre de l'algorithme d'analyse aboutit ainsi à des scores encourageants. Finalement, la robustesse des concepts exposés pourrait être estimée en menant une expérimentation plus large, intégrant un traitement complet de plusieurs cas réels et des formats de sortie variés.

Chapitre 5

Vers une théorie d'interface prenant en compte les propriétés d'états

Le raisonnement par contrat [88, 89] est une méthode puissante pour la conception de logiciels. Elle est accompagnée de plusieurs langages de programmation [15]. Ceux-ci sont notamment inspirés de la logique de Hoare [90] et de ses extensions, telles que la logique de séparation [91]. Dans le contexte de la conception de systèmes réactifs, un cadre formel puissant est offert par la théorie des contrats [4]. En tant que théorie générique, celle-ci soutient une grande variété de formalismes et de processus de conception, portant en commun une algèbre de composition et un concept de raffinement. Ces caractéristiques reflètent la décomposition d'exigences d'un système en plusieurs points de vue et composants.

Suivant ces principes, nous proposons de fonder une théorie autour d'un formalisme adapté pour la spécification de systèmes réactifs en temps discret. L'enjeu est la prise en charge d'une approche compositionnelle pour l'analyse d'exigences comportementales à un stade précoce du processus de conception. Il s'agit de répondre à différents besoins de l'ingénierie des exigences, en apportant des garanties de cohérences sur les documents de spécification.

Précisément, la mise en œuvre de systèmes complexes implique l'intégration de plusieurs composants, dont les entrées et sorties sont reliées à d'autres composants ou au monde extérieur. L'environnement d'un composant est constitué d'un ensemble d'entités qui n'est pas complètement connu lors du développement, ce qui implique de définir clairement le contexte d'utilisation. D'un autre côté, les comportements d'un système peuvent être observés selon deux dimensions : l'interaction avec l'environnement et l'évolution des états du système. La première dimension est traduite par une séquence d'événements, tandis que la seconde se reflète à l'aide de propriétés d'états.

Les théories d'interfaces visent à fournir une spécification fusionnée du comportement attendu d'un composant en cours de conception et des environnements possibles dans lesquels il peut fonctionner. Les exemples typiques de théories d'interface sont les *Automates d'interface* [3] et les *Interfaces modales* [13, 14, 4]. Ces formalismes de la théorie des automates, inspirés des automates d'entrée/sortie de Lynch [92, 93], sont capables de saisir à la fois la variabilité des conceptions possibles et les incertitudes concernant les environnements possibles d'un composant. La compatibilité des composants est également saisie en caractérisant les environnements dans lesquels des réalisations arbitraires des interfaces peuvent être correctement composées. Les théories d'interfaces peuvent également intégrer les machines de Moore (interfaces de Moore synchrones [94]).

La conception basée sur une théorie d'interface, ou, de manière plus générale, sur une théorie de contrat, se fonde sur un cadre dont la description est finie et des relations fondamentales calculables. En premier lieu, la relation entre une interface (ou un contrat) et une réalisation doit être formellement établie. De plus, ce cadre doit permettre de caractériser des opérateurs de raffinement et de composition (conjonction, produit et quotient). Le raffinement se rapporte aux flux verticaux, entre différents couches d'abstraction. A contrario, les opérateurs de composition agissent sur des flux horizontaux, au même niveau d'abstraction.

Ces approches sont appropriées pour accompagner la conception de systèmes complexes. Elles permettent d'exprimer la variabilité des conceptions envisageables et les incertitudes concernant les environnements possibles. En outre, la relation entre entrées/sorties de composants en fonction des événements observés est bien traitée. Néanmoins, les comportements représentés ne sont pas directement reliés aux états du composant, ce qui en limite fortement l'usage en pratique. De ce fait, les exigences issues de cahiers des charges, qui déterminent précisément des relations entre comportements possibles et états d'un système, ne peuvent être directement et complètement spécifiées à l'aide des formalismes cités. Cette limite restreint fortement l'intégration dans un processus automatisé, comme nous l'avons vu dans le chapitre 2.

Dans de nombreux cas, il serait donc souhaitable de relier ces comportements à l'état du composant. C'est ce que nous cherchons à faire dans la suite. Précisément, l'objectif est d'élaborer une théorie d'interfaces traitant les comportements observés selon les deux dimensions voulues, à savoir l'interaction avec l'environnement et l'état d'un système.

Plusieurs concepts fondamentaux sont introduits dans ce chapitre. Les modèles de réalisation, discutés dans la section 5.1, constituent un premier socle nécessaire à l'éla-

boration d'une théorie d'interface. Nous proposons d'exploiter la sémantique de Kripke pour spécifier le comportement du système avec des propriétés d'état. Les propriétés algébriques définies dans la section 5.2 complètent ce socle. Finalement, la section 5.3 porte sur la recherche d'un formalisme de spécification associable au modèle de réalisation ciblé (modèle de Kripke), et offrant un bon compromis en termes d'expressivité et de propriétés algébriques.

5.1 Modélisation de systèmes

L'élaboration d'une théorie d'interface implique la formalisation de deux concepts fondamentaux : les modèles de réalisation et les modèles de spécification. Les modèles de réalisation sont utilisés pour représenter les comportements possibles d'un composant donné. *A contrario*, les modèles de spécification doivent permettre de saisir des règles de comportements, en autorisant une certaine variabilité de réalisations possibles. Ces deux formalismes sont formellement reliés en définissant une relation de satisfaction.

La définition du comportement attendu d'un composant peut se traduire sous la forme d'un automate étiqueté, où les arcs définissent des transitions entre les différents états du système. L'étiquette associée à une transition définit alors un événement du système ou de son environnement.

La sémantique de Kripke, basée sur un ensemble de mondes possibles, est intéressante pour spécifier le comportement du système avec des propriétés d'état. Chaque monde peut être relié à un état du système vérifiant certaines propriétés définies comme des propositions, et les transitions entre les mondes sont les transitions entre les états. Il est également possible de définir des actions sous forme de propositions, où la proposition p_a signifierait que l'action a est exécutée. Avec cela, il est possible de spécifier le comportement du système avec des propriétés d'état et des séquences d'actions.

5.1.1 Syntaxe

Les propriétés d'état sont caractérisées par la valuation d'un ensemble fini de variables booléennes, ou propositions atomiques, que nous notons Π.

Définition 10 *L'alphabet* Π *est un ensemble de propositions atomiques. L'ensemble des parties de* Π *et l'ensemble des formules propositionnelles sur* Π *sont désignés respectivement par* 2^{Π} *et* $\boldsymbol{Prop}(\Pi)$.

Nous proposons de représenter le comportement d'un système avec un modèle de Kripke non déterministe. Cette structure est définie par un ensemble d'états et un ensemble de relations entre ces états. Chaque état est associé à une valuation de propositions atomiques (sous la forme d'un ensemble de propositions appartenant à l'alphabet considéré, désigné par Π).

Définition 11 *Modèle de Kripke (KM)*
Un modèle sur l'alphabet Π *est un tuple* $M = (\Pi, Q, Q_0, \rightarrow, \upsilon)$ *tel que :*

- Π *est un ensemble de propositions atomiques ;*
- Q *est un ensemble d'états ;*
- $Q_0 \subseteq Q$ *est un ensemble d'états initiaux ;*
- $\rightarrow \subseteq Q \times Q$ *est la relation de transition entre les états ;*
- $\upsilon : Q \mapsto 2^{\Pi}$ *est la fonction de valuation.*

Exemple 26 *La figure 5.1 modélise le comportement simple d'un ascenseur à trois étages. Nous considérons trois situations possibles : l'ascenseur est en mouvement (proposition m), l'ascenseur est arrêté à l'étage i (propositions* s_0, s_1, s_2, s_3*), la porte de l'ascenseur est ouverte à l'étage i (propositions* o_0, o_1, o_2, o_3*). On peut représenter toutes les propriétés par l'alphabet* $\Pi = \{m, s_0, .., s_3, o_0, .., o_3\}$*. Cette représentation montre que l'ascenseur doit se déplacer d'un étage à l'autre, et que les portes de l'ascenseur ne peuvent être ouvertes que si l'ascenseur est arrêté à l'étage correspondant.*

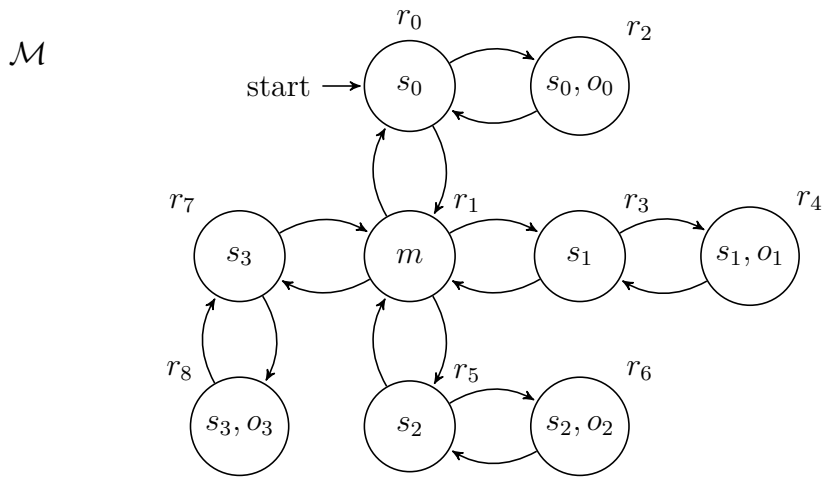


FIGURE 5.1 – Modèle pour un ascenseur à trois étages

La définition 12 apporte une distinction entre les modèles suivant l'alphabet dont ils dépendent.

Définition 12 *Modèles homogènes, modèles hétérogènes*
Deux modèles sont homogènes s'ils se rapportent au même alphabet. Ils sont hétérogènes s'ils se rapportent à deux alphabets distincts.

Cette distinction n'est pas fondamentale à ce stade : les définitions et les théorèmes seront généralement présentés pour des structures homogènes, mais ils peuvent être étendus à des structures hétérogènes. Nous utilisons également la relation de satisfaction telle qu'elle est habituellement définie : $\sigma \models \varphi$ si la valuation σ satisfait la formule φ.

Exemple 27 *Nous considérons un système représentant une navette. L'accent est mis sur les portes d'accès au véhicule. L'alphabet des propositions atomiques est* $\Pi = \{m, r, o\}$*, où* (o) *indique l'ouverture des portes,* (m) *que la navette est en mouvement, et* (r) *qu'une demande d'arrêt est en cours. La figure 5.2 montre une modélisation, sous la forme d'un modèle de Kripke, pour le comportement d'une porte. On suppose que le véhicule est initialement arrêté (*q_0*). Il peut se déplacer (*q_1*), et dans ce cas, un arrêt peut encore être demandé (*q_2*). La navette doit s'arrêter (*q_3*) pour que les portes soient ouvertes (*q_4*). Seules les propositions vraies sont notées dans les états. Les boucles ne sont pas représentées (elles sont implicites pour tous les états).*

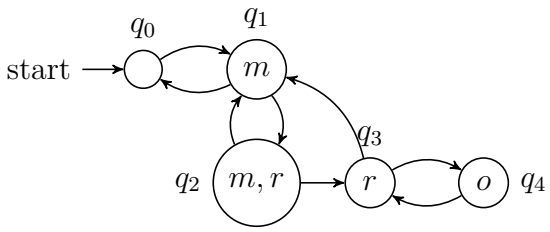


FIGURE 5.2 – Modélisation d'un système représentant une navette

Nous envisageons des spécifications modulaires. Pour une réalisation donnée, l'environnement peut évoluer sans aucune évolution observable au niveau du système. En fait, il faut considérer qu'un système peut systématiquement rester dans le même état. Cette caractéristique est connue sous le nom de *stuttering invariance* [95] (invariance par bégaiement). Elle est importante pour définir certaines propriétés ou opérations, en particulier la composition et la conjonction. Aussi, nous supposons que tout modèle vérifie la propriété d'invariance par bégaiement.

Définition 13 *Invariance par bégaiement (Stuttering Invariant)*
Un modèle est invariant au bégaiement si $\forall q \in Q, q \rightarrow q$.

Remarque 1 *Boucle implicite*
Les boucles sont implicites. Elles n'apparaissent pas nécessairement dans les représentations graphiques. L'ensemble des transitions ($\rightarrow$) *peut facilement être étendu pour inclure ces boucles.*

5.1.2 Sémantique

La notion de trace est introduite ci-dessous pour caractériser les exécutions d'un modèle, et pour étendre la fonction de valuation définie pour les états du modèle aux mots d'un langage.

Définition 14 *Trace de modèle*
Soit Π *un ensemble de propositions atomiques, et* $\mathcal{M}$ *un modèle de Kripke sur* Π.
Une trace u *est une suite finie de valuations sur* 2^{Π}. *La concaténation de deux traces* u *et* v *définit une nouvelle trace, dénommée* $u.v$. *On écrit* ϵ *la trace vide.*
On écrit $(2^{\Pi})^*$ *l'ensemble des traces sur l'alphabet* Π.

Définition 15 *Fonction de transition étendue aux mots*
Soit Π *un ensemble de propositions atomiques, et* $\mathcal{M}$ *un modèle de Kripke sur* Π.
La fonction de transition étendue aux mots sur $(2^{\Pi})^*$ *est la fonction* $\Delta : 2^{Q} \times (2^{\Pi})^* \rightarrow 2^{Q}$ *telle que* $\forall P \subseteq Q$ *:*

- $\Delta(P, \epsilon) = P$,
- $\forall u.a \in (2^{\Pi})^*, a \in 2^{\Pi}, \Delta(P, u.a) = \{q' | \exists q \in \Delta(P, u)$ *tel que* $q \rightarrow q'$ *et* $\upsilon(q') = a\}$.

Nous pouvons finalement définir le langage support d'un modèle, reflétant toutes les exécutions possibles.

Définition 16 *Langage support*
Soit $\mathcal{M}$ *un modèle. Le langage support de* $\mathcal{M}$ *est* $\mathcal{L}(\mathcal{M}) = \{u | \Delta(Q_0, u) \neq \emptyset\}$.

Remarque 2 *Chemin associé aux mots*
Soit $\mathcal{M}$ *un modèle. Soit* u *un mot appartenant à* $\mathcal{L}(\mathcal{M})$. *Le mot* u *définit une suite de valuations de* $\mathcal{M}$, *noté* $\sigma_0, \sigma_1, ..., \sigma_{m-1}, \sigma_m$, *et il est possible d'associer un chemin* $q_0, q_1, ...q_{m-1}, q_m$ *à* u *tel que :*

— $q_0 \in Q_0$ *et* $\upsilon(q_0) = \sigma_0$,

— $\forall i < m, q_i \rightarrow q_{i+1} \wedge \upsilon(q_i + 1) = \sigma_{i+1}$.

Cette propriété est une conséquence directe des définitions 15 et 16. Les modèles pouvant être non déterministes, ce chemin n'est pas nécessairement unique.

5.1.3 Produit de modèles

La composition parallèle, également appelée produit, est une opération importante pour définir le comportement d'un système composé de plusieurs composants. La définition de la composition parallèle pour les KM utilise un opérateur de permutation, désigné par $\leftrightarrow_{23}$, défini préalablement.

Définition 17 *Permutation* $\leftrightarrows_{23}$
Soit q_1, q_1', q_2, q_2' *quatre états, et* (q_1, q_1', q_2, q_2') *un tuple. La permutation de l'état* 2 *et de l'état* 3 *pour le tuple* (q_1, q_1', q_2, q_2')*, désigné par* $\leftrightarrows_{23}$*, est le tuple* (q_1, q_2, q_1', q_2').

Définition 18 *Produit de modèles homogènes*
Soit $M_1 = (\Pi_1, Q_1, Q_{0,1}, \rightarrow_1, \upsilon_1)$ *et* $M_2 = (\Pi_2, Q_2, Q_{0,2}, \rightarrow_2, \upsilon_2)$ *deux modèles, et l'alphabet* $\Pi_{1,2} = \Pi_1 \cap \Pi_2$.
Le produit de M_1 *et* M_2*, noté* $M_1 \times M_2$*, est le modèle* $M = (\Pi, Q, Q_0, \rightarrow, \upsilon)$ *tel que :*

— $Q = \{(q_1, q_2) | (\upsilon_1(q_1) \cap \Pi_{1,2}) = (\upsilon_2(q_2) \cap \Pi_{1,2})\}$

— $Q_0 = (Q_{0,1} \times Q_{0,2}) \cap Q$

— $\rightarrow = \leftrightarrows_{23} (\rightarrow_1 \times \rightarrow_2) \cap Q^2$

— $\upsilon(q_1, q_2) = \upsilon_1(q_1) \cup \upsilon_2(q_2)$

Remarque 3 *Le produit de modèles homogènes est équivalent au produit synchrone.*

Propriété 2 *Le produit des modèles homogènes satisfait aux propriétés suivantes, à isomorphisme près, par permutation des composantes des états :*

— *commutativité :* $M_1 \times M_2 = M_2 \times M_1$,

— *associativité :* $M_1 \times (M_2 \times M_3) = (M_1 \times M_2) \times M_3$,

— *élément neutre :* $M_e = \{\{q\}, \{q\}, \rightarrow, \upsilon\}$*, avec* $\rightarrow = (q, q)$ *et* $\upsilon(q) = \emptyset$.

Preuve 2 *Soit* $M_1 = (\Pi_1, Q_1, Q_{0,1}, \rightarrow_1, \upsilon_1)$ *et* $M_2 = (\Pi_2, Q_2, Q_{0,2}, \rightarrow_2, \upsilon_2)$ *deux modèles, et l'alphabet* $\Pi_{1,2} = \Pi_1 \cap \Pi_2$.

Pour tous les états $q_1 \in Q_1$, $q_2 \in Q_2$, *nous vérifions* $(\upsilon_1(q_1) \cap \Pi_{1,2}) = (\upsilon_2(q_2) \cap \Pi_{1,2}) \Leftrightarrow \upsilon_2(q_2) \cap \Pi_{1,2}) = (\upsilon_1(q_1) \cap \Pi_{1,2})$. *On peut donc définir l'application bijective* $f : Q_1 \times Q_2 \rightarrow Q_2 \times Q1$ *telle que* $f((q_1, q_2)) = (q_2, q_1)$. *Cette application définit un isomorphisme entre les ensembles* Q *de* $M_1 \times M_2$ *et* $M_2 \times M_1$. *Le même raisonnement s'applique pour* Q_0, $\rightarrow$ *and* $\upsilon(q_1, q_2)$. *Nous vérifions ainsi la commutativité du produit modèle.*

Pour tous les états $q_1 \in Q_1$, $q_2 \in Q_2$, $q_3 \in Q_3$, *on vérifie l'associativité des égalités entre* $(\upsilon_1(q_1) \cap \Pi_{1,2})$, $(\upsilon_2(q_2) \cap \Pi_{1,2})$ *et* $(\upsilon_3(q_3) \cap \Pi_{1,2})$. *Nous pouvons donc définir l'application bijective* $f : Q_1 \times (Q_2 \times Q_3) \rightarrow (Q_1 \times Q_2) \times Q_3$ *telle que* $f((q_1, (q_2, q_3)) = ((q_1, q_2), q_3)$. *Cette application définit un isomorphisme entre les ensembles* Q *de* $M_1 \times (M_2 \times M_3)$ *et* $(M_1 \times M_2) \times M_3$. *Le même raisonnement s'applique à* Q_0, $\rightarrow$ *et* $\upsilon(q_1, \upsilon(q_2, q_3))$. *Nous vérifions ainsi l'associativité du produit modèle.*

Le modèle M_e *ne contient qu'un seul état* $q \in Q$. *Nous pouvons donc définir la correspondance bijective* $f : Q_1 \times \{q_e\} \rightarrow Q1$ *telle que* $f((q_1, q_e)) = q_1$. *Cette application définit un isomorphisme entre les ensembles* Q *de* $M_1 \times M_e$ *et* Q_1. *Le même raisonnement s'applique à* Q_0, $\rightarrow$ *et* $\upsilon(q_1, q_e)$. *Nous montrons ainsi que* M_e *est un élément neutre pour le produit modèle.*

5.2 Algèbre de composition

La théorie de contrats [4] peut être vue comme un ensemble de directives méthodologiques. Elle se caractérise par différentes opérations applicables pour assister la conception de systèmes. Ces opérations sont utilisées pour accompagner la réutilisation des composants, en soutenant les systèmes ouverts, dans une démarche maîtrisée. Elles permettent également de gérer et fusionner les exigences de différents points de vue, et autorise ainsi le développement indépendant de composants. Cette section propose une description des différentes opérations qui permettent d'établir une algèbre de composition complète.

Dans la suite, la notation CC s'applique au cahier des charges de base, vu comme une composition d'exigences (contrats). Les spécifications formelles, reflétant chacune des exigences de la spécification, seront dénotées $\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_n$, et la composition de ces spécifications sera désignée par $\mathcal{S}$. Nous considérons CC comme équivalent de $\mathcal{S}$, soit $CC = \bigwedge_i \mathcal{S}_i$. Finalement, la notation $\mathcal{M}$ est utilisée pour les modèles de réalisation.

5.2.1 Consistance et relation de satisfaction

Une exigence peut être vue comme un contrat entre un système, ou un composant, et son environnement. Elle spécifie des obligations en fonction de contraintes opérationnelles. Dès lors, un cahier des charges peut se ramener à une composition de contrats, en apportant une séparation claire des responsabilités entre les différents acteurs de la chaîne de développement. La notion de consistance traduit l'absence attendue d'incohérences entre les exigences spécifiées. Nous pouvons établir qu'un cahier des charges CC est consistant si, et seulement si, il existe au moins une réalisation qui satisfait toutes les spécifications associées, c'est-à-dire s'il existe $\mathcal{M} \models \mathcal{S}_1, ..., \mathcal{S}_n$.

Définition 19 *Une spécification $\mathcal{S}$ est consistante si, et seulement si, il existe un modèle $\mathcal{M}$ qui satisfait $\mathcal{S}$, c'est à dire s'il existe $\mathcal{M} \models \mathcal{S}$.*

5.2.2 Relation de raffinement

Le concept de raffinement est introduit pour assurer la compatibilité d'implémentations indépendantes, à différents niveaux d'abstraction. Ce concept permet de prendre en considération l'évolution des exigences d'un cahier des charges, et rend possible la substitution de composants de manière sûre.

Définition 20 *$\mathcal{S}_1$ est un raffinement de $\mathcal{S}_2$, noté $\mathcal{S}_1 \preceq \mathcal{S}_2$, si, et seulement si, pour tout modèle $\mathcal{M}$, $\mathcal{M} \models \mathcal{S}_1$ implique que $\mathcal{M} \models \mathcal{S}_2$.*

5.2.3 Conjonction

Les cahiers des charges peuvent être structurés autour de différents points de vue portant sur un même système, ou une partie de celui-ci. Ceux-ci se reflètent dans les exigences des documents de spécification, dont l'interprétation est la conjonction de toutes ces implications. Nous pouvons établir que $\mathcal{S}' = \mathcal{S}'_1, ..., \mathcal{S}'_n$ est un raffinement consistant de $\mathcal{S}$ si, et seulement si, toutes les réalisations qui satisfont à l'intégration conjointe de $\mathcal{S}'_1, ..., \mathcal{S}'_n$ sont des réalisations de $\bigwedge_i \mathcal{S}_i$ (correspondant à la spécification $\mathcal{S}$).

Définition 21 *La conjonction de $\mathcal{S}_1$ et $\mathcal{S}_2$, notée $\mathcal{S}_1 \wedge \mathcal{S}_2$, est la plus petite spécification $\mathcal{S}$ telle que tout modèle $\mathcal{M}$ de $\mathcal{S}_1$ et $\mathcal{S}_2$ est un modèle de $\mathcal{S}$, c'est à dire : $\mathcal{S}_1 \wedge \mathcal{S}_2 = min\{\mathcal{S} \mid \forall \mathcal{M}, \mathcal{M} \models \mathcal{S}_1$ and $\mathcal{M} \models \mathcal{S}_2 \Rightarrow \mathcal{M} \models \mathcal{S}\}$.*

L'opération de conjonction formalise la notion d'ensemble d'exigences. Elle peut être reliée aux notions de consistance et de raffinement. Il est ainsi possible de garantir formellement la consistance d'un document de spécification en vérifiant que la conjonction des spécifications formelles reflétant chacune des exigences est satisfiable, c'est à dire qu'il existe au moins un modèle satisfaisant la spécification obtenue par conjonction.

5.2.4 Composition parallèle

L'approche classique pour le développement de systèmes complexes consiste à proposer une architecture globale, avec une décomposition en un assemblage de composants, ou sous-systèmes. Chaque fournisseur peut être amené à travailler sur une partie spécifique. Dans cette optique, une spécification principale $\mathcal{S}$ est divisée en plusieurs sous-spécifications, $\mathcal{S}'_1, ..., \mathcal{S}'_n$, dont le développement peut être indépendant. Pour chacun de ces sous-systèmes, une spécification formelle peut être définie. L'intégration des différents composants, respectant chacune des spécifications, aboutit à la composition des sous-systèmes définissant l'architecture principale. Celle-ci se traduit par la composition parallèle des spécifications. L'intégration conjointe des composants renvoie également vers cette opération.

Définition 22 *La composition parallèle de $\mathcal{S}_1$ et $\mathcal{S}_1$, notée $\mathcal{S}_1 \otimes \mathcal{S}_2$, est la plus petite spécification $\mathcal{S}$ telle que , pour tout modèle $\mathcal{M}_1$ de $\mathcal{S}_1$ et tout modèle $\mathcal{M}_2$ de $\mathcal{S}_2$, le produit $\mathcal{M}_1 \times \mathcal{M}_2$ est un modèle de $\mathcal{S}$, c'est à dire : $\mathcal{S}_1 \otimes \mathcal{S}_2 = min\{\mathcal{S} \mid \forall M_1 \models \mathcal{S}_1, \forall M_2 \models \mathcal{S}_2, M_1 \times M_2 \models \mathcal{S}\}$.*

5.2.5 Quotient

Le quotient s'inscrit également dans un cadre compositionnel. Cette opération permet de sécuriser la réutilisation de composants, provenant par exemple d'une bibliothèque spécifique. Précisément, le quotient est l'adjoint à droite de l'opérateur du produit. Formellement, le quotient de $\mathcal{S}_1$ par $\mathcal{S}_2$ caractérise les réalisations $\mathcal{M}$ tel que toute réalisation de $\mathcal{S}_2$ composée avec $\mathcal{M}$ est une réalisation de $\mathcal{S}_1$.

Définition 23 $\mathcal{S}_1/\mathcal{S}_2 = max\{\mathcal{S} \mid \mathcal{S}_2 \times \mathcal{S} \leq \mathcal{S}\}$.

5.3 Spécifications

Partant des exigences d'un cahier des charges, notre démarche vise la fondation d'une théorie complète accompagnant l'ingénieur à un stade précoce du processus de conception. Dans cette optique, nous cherchons à définir un formalisme reliant les comportements possibles ou interdits aux états d'un système. Notre proposition tourne autour des théories d'interface, que nous projetons d'étendre pour prendre en considération l'évolution des états d'un système.

En effet, comme nous l'avons vu dans le chapitre 2, les automates classiques ne permettent pas d'exprimer la variabilité de réalisations, les automates d'interface [3] n'ont pas l'opération de conjonction, tandis que les logiques temporelles [1, 2] n'ont pas la composition parallèle. Ces propriétés sont couvertes pour le μ-calcul modal [96, 97], au prix d'une complexité de calcul rédhibitoire. Les spécifications modales [51] représentent un premier compromis possible.

5.3.1 Spécifications modales

Concrètement, les spécifications modales prennent la forme d'automates dont les transitions sont typées avec les modalités *must* et *may*. Ils définissent ainsi un ensemble de modèles. Une transition de type *must* est obligatoirement présente dans chaque composant qui implémente la spécification modale, tandis qu'une transition de type *may* est facultative.

Suivant la définition de J.-B. Raclet [51], une spécification modale est un tuple $S = (Q, q_0, \Sigma, \Delta^m, \Delta^M)$. Q est un ensemble fini d'états. q_0 est l'unique état initial. Σ est un ensemble fini d'actions. $\Delta^m \subseteq Q \times \Sigma \times Q$ est l'ensemble des transitions facultatives ; $\Delta^M \subseteq Q \times \Sigma \times Q$ est l'ensemble des transitions obligatoires. Enfin, la condition de consistance suivante doit être respectée : $\Delta^M \subseteq \Delta^m$ et $q_0 \in Q$. Cette condition signifie que chaque transition obligatoire (appartenant à l'ensemble Δ^M) est également une transition facultative (appartenant à l'ensemble Δ^m).

Le modèle d'une spécification modale est défini à partir de la relation de satisfaction, notée $\models$. Soit $S = (Q, q_0, \Sigma, \Delta^m, \Delta^M)$ une spécification modale. Soit $M = (R, r_0, \Sigma, \Gamma \subseteq R \times \Sigma \times R)$ avec Γ déterministe. M est un modèle de S, noté $M \models S$, si, et seulement si, il existe $\rho \subseteq R \times Q$ tel que :

1. $(r_0, q_0) \in \rho$
2. $\forall (r, q) \in \rho, \forall \sigma \in \Sigma$

(a) $\forall r' \in R, (r, \sigma, r') \in \Gamma \Rightarrow \exists q' \in Q$ tel que $(q, \sigma, q') \in \Delta^m$ et $(r', q') \in \rho$

(b) $\forall q' \in Q, (q, \sigma, q') \in \Delta^M \Rightarrow \exists r' \in R$ tel que $(r, \sigma, r') \in \Gamma$

La figure 5.3 montre une spécification modale (à gauche) et deux modèles possibles de cette spécification (à droite), où a et b sont deux actions (ou événements). La présence d'un arc continu signifie que la transition est obligatoire ("must"), la présence d'un arc en pointillé que la transition est facultative ("may") et l'absence d'arc indique que la transition est interdite dans toute réalisation.

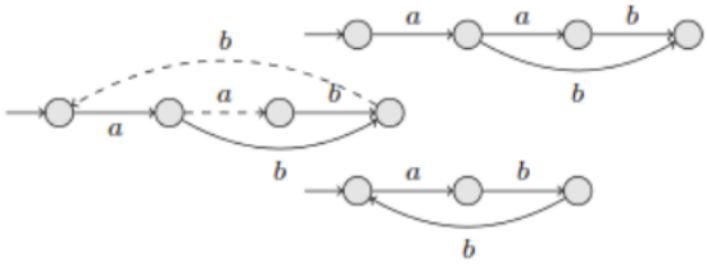


FIGURE 5.3 – Une spécification modale et deux modèles possibles

Les spécifications modales intègrent les opérations de conjonction, de composition et de quotient avec une complexité algorithmique polynomiale [13]. Les caractéristiques algébriques de ces opérations peuvent être précisées à partir de la définition du raffinement. Une spécification modale S_1 est le raffinement d'une spécification S_2 si, et seulement si, les modèles de S_1 sont aussi des modèles de S_2. Formellement, le raffinement, noté $\preceq$, se définit comme suit : $S_1 \preceq S_2$ si, et seulement si, pour tout modèle M, nous avons $M \models S_1 \Rightarrow M \models S_2$. Nous pouvons finalement définir la conjonction, la composition parallèle et le quotient :

— $S_1 \otimes S_2 = min\{S \mid \forall M_1 \models S_1, \forall M_2 \models S_2, M_1 \times M_2 \models S\}$

— $S_1 \wedge S_2 = min\{S \mid \forall M, M \models S_1 et M \models S_2 \Rightarrow M \models S\}$

— $S_1 / S_2 = max\{S \mid S_2 \otimes S \leq S_1\}$

De fait, les spécifications modales conviennent parfaitement à la représentation d'interfaces de différents composants d'un système. Certes, les spécifications modales sont strictement moins expressives que le μ-calcul modal. Cette limitation est néanmoins acceptable pour formaliser les propriétés de sécurité des spécifications des systèmes cyber-physiques.

5.3.2 Structure de spécification pour les modèles de Kripke

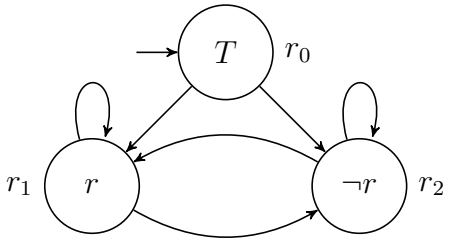
Les interfaces modales combinent de bonnes propriétés algébriques et une complexité algorithmique raisonnable. Cependant, seul le comportement d'entrée/sortie des composants réactifs est traité, alors que, dans de nombreux cas, il serait souhaitable de relier ce comportement à l'état du composant. L'utilisation d'une structure de Kripke à la place d'un automate classique répond à cette contrainte pour les modèles de réalisation. Dès lors, quel formalisme pourrait-on associer à ce modèle ? De fait, les interfaces modales n'admettent que des systèmes de transition finis.

Les automates moniteurs [9, 10], étudiés en détail au chapitre 7, sont proposés en première intention. Ils permettent de saisir les propriétés d'état, exprimées sous forme de formules de proposition, et les propriétés temporelles, exprimées sous forme de séquences d'actions obligatoires ou interdites. À partir d'un état initial, un moniteur peut observer le fonctionnement d'un modèle, et le valider s'il ne passe pas par un état illégal. Une théorie de contrats basée sur les moniteurs comme spécification peut être établie. Dans ce cadre, les réalisations sont des modèles de Kripke. Comme nous le verrons, il est possible de doter cette théorie d'une algèbre partielle. La définition des opérations de composition est néanmoins plus difficile, sauf pour la conjonction.

L'utilisation des moniteurs comme socle d'une théorie d'interface n'est pas pleinement satisfaisante. Nous proposons d'utiliser des ensembles spéciaux, nommés ensembles d'acceptation, pour définir la variabilité des réalisations possibles à partir d'une situation donnée. Un nouveau formalisme peut ainsi être défini par extension des moniteurs. Les automates d'acceptation propositionnels et déterministes (*Deterministic Propositional Acceptance Automata*, DPAA) résultent de cette démarche. La caractéristique principale de ce formalisme est qu'il permet d'exprimer les comportements attendus lorsque le système est dans un état particulier. Comme les moniteurs, les DPAA combinent des propriétés d'état, exprimées sous forme de formules propositionnelles, et des propriétés temporelles discrètes, exprimées sous forme d'événements obligatoires ou interdits. Ils étendent les systèmes de transitions modaux en prenant comme modèles les structures de Kripke plutôt que des systèmes de transitions étiquetés. Ainsi, ce nouveau formalisme, caractérisé dans le chapitre 8, offre également la prise en charge de propriétés d'état, tout en étant doté de l'ensemble des propriétés algébriques attendues.

Concrètement, les DPAA capturent la variété des implémentations en les exprimant sous la forme d'ensembles d'acceptations. Les exécutions d'un modèle peuvent ainsi être observées à la manière d'un moniteur. Elles sont validées si les ensembles d'états succes-

seurs des états du modèle peuvent être correctement reliés aux ensembles d'acceptation des états de la spécification. Un exemple est donné dans la figure 5.4 qui définit les comportements autorisés des portes d'une navette en fonction d'une demande d'arrêt. Les propositions m, r et o indiquent respectivement le mouvement du bus, la requête d'arrêt et l'ouverture de la porte. La structure de Kripke de la figure 5.2 modélise une implémentation envisageable. Les ensembles d'acceptation associés à chaque état sont représentés dans le tableau associé.



	$Acc(r_i)$
r_0	$2^{2^{2^{\Pi}}}$
r_1	$\{\alpha \mid \forall \sigma \in \alpha, o \in \sigma\}$
r_2	$2^{2^{2^{\Pi}}}$

FIGURE 5.4 – DPAA d'une exigence spécifiant l'ouverture des portes d'une navette à la demande d'un passager.

5.3.3 Complément

Finalement, notre cheminement passe par l'étude de représentations finies permettant de caractériser un ensemble potentiellement infini de comportements possibles. Les ensembles d'acceptation sont considérés dans le chapitre suivant pour cette perspective (chapitre 6). Les structures envisagées, fusionnant implémentations et environnements, s'inscrivent dans le cadre des théories d'interface. Ces théories combinent modèles de réalisation et spécifications formelles. Les modèles permettent de définir les comportements possibles d'un composant spécifique, tandis qu'une spécification formelle définit un ensemble potentiellement infini de réalisations possibles. La sémantique de Kripke est l'option privilégiée comme modèle de réalisation. Les chapitres 7 et 8 apportent le complément à cette structure pour former deux théories d'interface, utilisant respectivement les automates moniteurs et les automates à ensembles d'acceptation propositionnels comme modèles de spécification.

Chapitre 6

Ensembles d'acceptation

Les ensembles d'acceptation (AS [1]) permettent de saisir avec une grande souplesse la variabilité de mise en œuvre d'une spécification. Ces ensembles seront exploités pour caractériser les formalismes de spécification des chapitres 7 et 8. Ainsi, les automates déterministes à ensembles d'acceptation propositionnels (DPAA [2]) sont des automates dont les états sont étiquetés avec des ensembles particuliers, appelés ensembles d'acceptation (AS). Dans ce chapitre, nous proposons d'étudier les propriétés algébriques de ces ensembles.

6.1 Définition

L'idée des ensembles d'acceptation pour le représentation de la variabilité de réalisations a été introduite dans la thèse de Jean-Baptiste Raclet [13]. Nous transposons ce formalisme aux propriétés d'états, vues comme propositions atomiques. Plus précisément, les ensembles d'acceptation portent sur les valuations booléennes d'un ensemble de propositions atomiques, que nous nommons labels.

Définition 24 *Ensemble d'acceptation (Acceptance Set, AS)*
Soit Π *un ensemble de propositions atomiques. Un label de* Π *est une valuation booléenne sur* Π*. Un ensemble d'acceptation Acc sur* Π *est un ensemble d'ensembles de labels sur* Π*, c'est-à-dire* $Acc \in 2^{2^\Sigma}$*, avec* $\Sigma = 2^\Pi$*.*

Les labels peuvent être reliés aux propriétés d'un système donné. Chaque $\alpha \in 2^\Sigma$, nommé *ready-set*, définit un variant sous la forme d'ensembles de propriétés qu'un système doit satisfaire. Un ensemble d'acceptation peut ainsi s'interpréter comme un ensemble de variants, exprimant ainsi la variabilité des réalisations possibles pour une spécification donnée (les spécifications admettant plusieurs réalisations en général).

1. AS : Acceptance Set
2. DPAA : Deterministic Propositional Acceptance Automata

6.2 Algèbre des ensembles d'acceptation

Une algèbre avec de bonnes propriétés est nécessaire pour que les ensembles d'acceptation soient utilisables pour définir une théorie d'interface [4]. L'algèbre consiste en une relation de satisfaction $\models$, une relation de raffinement $\preceq$ et des opérateurs de composition $\wedge$, $\otimes$ et $/$. Nous commençons par la définition d'une algèbre pour les AS homogènes, pour un alphabet uniforme de propositions atomiques Π. Cet algèbre est ensuite étendue aux AS hétérogènes.

6.2.1 Algèbre des AS homogènes

La relation de satisfaction est la sélection d'un *ready-set* parmi les éléments de l'AS. Le raffinement est l'inclusion d'un AS. La conjonction est l'intersection d'AS, et le produit est l'intersection point à point des *ready-sets*. Le quotient, repris de la thèse de J.B. Raclet [51], est l'adjoint droit de l'opérateur du produit : $X \otimes B \preceq A \Leftrightarrow X \preceq A/B$. Ces opérateurs définissent une théorie d'interface complète.

Définition 25 *Algèbre des AS homogènes*
Les opérateurs suivants composent une algèbre pour les AS homogènes :

- *satisfaction :* $\alpha \models Acc$ *ssi* $\alpha \in Acc$ *;*
- *raffinement :* $Acc_1 \preceq Acc_2$ *ssi* $Acc_1 \subseteq Acc_2$ *;*
- *conjonction :* $Acc_1 \ \wedge \ Acc_2 = Acc_1 \cap Acc_2$ *;*
- *produit :* $Acc_1 \otimes Acc_2 = \{\alpha_1 \cap \alpha_2 \mid \alpha_1 \in Acc_1, \alpha_2 \in Acc_2\}$ *;*
- *quotient :* $Acc_1/Acc_2 = \{\alpha \mid \forall \beta \in Acc_2, \alpha \cap \beta \in Acc_1\}$.

6.2.2 Algèbre des AS hétérogènes

Nous allons maintenant définir plusieurs opérateurs sur les labels. Sur la base de ces derniers, les opérateurs sur les ready-sets sont ensuite définis. Finalement, ceux-ci permettent d'étendre l'algèbre des ensembles d'acceptation homogènes aux ensembles d'acceptation hétérogènes.

Opérateurs sur les labels. Soit $\upsilon_1 \subseteq \Pi_1$ et $\upsilon_2 \subseteq \Pi_2$ deux valuations sur des alphabets hétérogènes : Π_1 et Π_2. On note Π_{12} l'intersection entre ces alphabets, c'est-à-dire $\Pi_{12} = \Pi_1 \cap \Pi_2$. L'opérateur $\bowtie$ est une relation de synchronisation telle que $\upsilon_1 \bowtie \upsilon_2$ si, et

seulement si, $v_1 \cap \Pi_{12} = v_2 \cap \Pi_{12}$. $v_1 \sqcup v_2 \subseteq (\Pi_1 \cup \Pi_2)$ est défini comme $v_1 \sqcup v_2 = v_1 \cup v_2$ si $v_1 \bowtie v_2$, non défini autrement. $v_1 \sqsubseteq v_2$ si, et seulement si $\Pi_1 \supseteq \Pi_2$ et $v_1 \bowtie v_2$.

Opérateurs pour ready-sets. Nous utilisons $\bowtie$, $\sqcup$ et $\sqsubseteq$ sur les labels pour définir $\bowtie$, $\sqcap$, $\sqsubseteq$ et $\downarrow$ sur les ready-sets : $\alpha_1 \bowtie \alpha_2$ si, et seulement si, $\forall v_1 \in \alpha_1, \exists v_2 \in \alpha_2, v_1 \bowtie v_2$ et $\forall v_2 \in \alpha_2, \exists v_1 \in \alpha_1, v_2 \bowtie v_1$; $\alpha_1 \sqcap \alpha_2 = \{v_1 \sqcup v_2 \mid v_1 \in \alpha_1, v_2 \in \alpha_2, v_1 \bowtie v_2\}$; $\alpha_1 \sqsubseteq \alpha_2$ si, et seulement si, il existe une surjection $f : \alpha_1 \to \alpha_2$ tel que $\forall v_1 \in \alpha_1, v_1 \sqsubseteq f(v_1)$. L'opérateur $\downarrow$ spécifie une restriction des ready-sets sur un ensemble de propositions atomiques, à savoir $\alpha_{\downarrow\Pi'} = \{\sigma' : \Pi' \mid \exists \sigma \in \alpha, \sigma \bowtie \sigma'\}$. Il est utilisé pour cacher les propositions atomiques.

La définition 25 est adaptée avec les opérateurs pour ready-sets. Dans la définition suivante, nous considérons Acc sur l'alphabet Π, Acc_1 et Acc_2 sur les alphabets Π_1 et Π_2 tels que $\Pi_2 \subseteq \Pi_1$, et un ready-set $\alpha \subseteq 2^{2^{\Pi_\alpha}}$, tel que $\Pi \subseteq \Pi_\alpha$.

Définition 26 *Algèbre des AS hétérogènes*

Les opérateurs suivants composent une algèbre pour les AS hétérogènes :

- *satisfaction :* $\alpha \models Acc$ *ssi* $\alpha_{\downarrow\Pi} \in Acc$ *;*
- *raffinement :* $Acc_1 \preceq Acc_2$ *ssi* $\forall \alpha_1 \in Acc_1, \alpha_1 \models Acc_2$ *;*
- *conjonction :* $Acc_1 \ \wedge \ Acc_2 = \{\alpha_1 \sqcap \alpha_2 \mid \alpha_1 \in Acc_1, \alpha_2 \in Acc_2, \alpha_1 \bowtie \alpha_2\}$ *;*
- *produit :* $Acc_1 \otimes Acc_2 = \{\alpha_1 \sqcap \alpha_2 \mid \alpha_1 \in Acc_1, \alpha_2 \in Acc_2\}$ *;*
- *quotient :* $Acc_1 / Acc_2 = \{\alpha \subseteq 2^{\Pi_1 \cup \Pi_2} \mid \forall \beta \in Acc_2, (\alpha \sqcap \beta)_{\downarrow\Pi_1} \in Acc_1\}$.

Exemple 28 *La figure 6.1 illustre l'évaluation de la conjonction en appliquant cette méthode.*

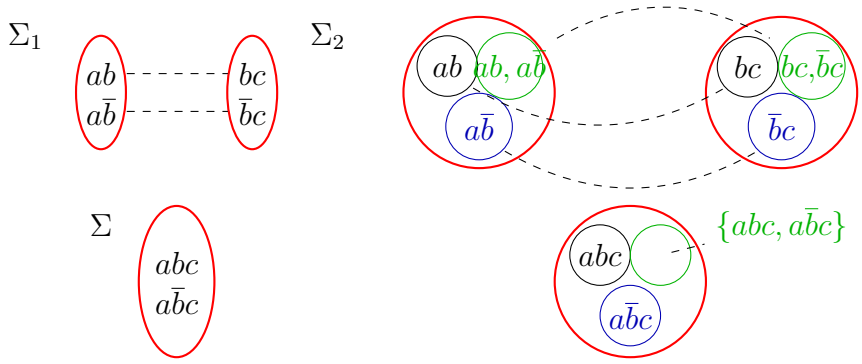


FIGURE 6.1 – Exemple pour la conjonction (synchronisation des labels à gauche, synchronisation des ready-sets à droite, résultat en bas)

Remarque 4 *On retrouve l'algèbre pour les ensembles d'acceptation homogènes lorsque ces définitions sont appliquées à des alphabets homogènes, à savoir lorsque* $\Pi_1 = \Pi_2$.

6.3 Représentation implicite

La manipulation explicite des AS implique de construire des ensembles de plus en plus grands avec une complexité de calcul potentiellement rédhibitoire. Par exemple, considérons deux alphabets disjoints Π_a et Π_b de 10 éléments chacun, avec $a \in \Pi_a$ et $b \in \Pi_b$. L'union de ces deux alphabets contient 20 éléments. Considérons deux AS $A = \{\{a\}\}$ et $B = \{\{b\}\}$. Le quotient de ces deux ensembles donne un ensemble d'acceptation très grand, dont les ready-sets correspondent à tous les comportements possibles à condition qu'il contienne $\sigma = \{ab\}$ et non $\sigma' = \{\overline{a}b\}$, c'est-à-dire $Acc_1/Acc_2 = \{\alpha \subseteq 2^{\Pi_1 \cup \Pi_2} \mid (\exists \sigma \subseteq \Pi_1 \cup \Pi_2$ tel que $ab \bowtie \sigma$ et $\sigma \in \alpha)$ et $(\forall \sigma : \Pi_1 \cup \Pi_2, \overline{a}b \bowtie \sigma \Rightarrow \sigma \notin \alpha)\}$. La taille de cet ensemble est d'environ $2^{2^{18}}$.

Représentation implicite. Cela justifie la recherche d'une représentation implicite, en utilisant des fonctions caractéristiques représentant des ready-sets. Comme nous souhaitons également éviter d'énumérer les ready-sets qui appartiennent à un AS, nous proposons d'utiliser un ensemble de paramètres pour énumérer les ready-sets. Les labels sur l'ensemble des propositions atomiques Π sont définis comme des fonctions booléennes $\overrightarrow{\pi} : \Pi \to \mathbb{B}$. De la même manière, les valeurs des paramètres sont codées, en utilisant un ensemble de variables booléennes $\mathcal{P} = \{p_1, ..., p_m\}$. Ainsi, les valeurs des paramètres sont des fonctions booléennes $\overrightarrow{p} : \mathcal{P} \to \mathbb{B}$. Cette représentation peut être très concise dans les bons cas, en utilisant par exemple des diagrammes de décision binaires.

Définition 27 *Représentation implicite des AS*
La représentation implicite d'un ensemble d'acceptation Acc est un tuple $(\Pi, \mathcal{P}, \psi, \phi)$ *tel que :*

- Π *est un alphabet de propositions atomiques,*
- $\mathcal{P}$ *est un ensemble de paramètres,*
- $\psi : (\mathcal{P} \to \mathbb{B}) \to \mathbb{B}$ *est la fonction caractéristiques des paramètres,*
- $\phi : (\mathcal{P} \to \mathbb{B}) \to (\Pi \to \mathbb{B}) \to \mathbb{B}$ *est la fonction caractéristique des ready-sets.*

Intuitivement, la fonction ψ est utilisée pour spécifier quelles valeurs de paramètres sont significatives, tandis que la fonction ϕ est utilisée pour définir le contenu d'un ensemble de valeurs de paramètres données. Par exemple, l'ensemble d'acceptation $Acc = \{\{a\}, \{a, ab\}, \{a, ac\}\}$ sur l'alphabet $\Pi = \{a, b, c\}$ peut être implicitement défini avec deux paramètres ($\mathcal{P} = \{p, q\}$), et les fonctions caractéristiques $\psi = \neg p \vee \neg q$ et $\phi = \sigma_1 \vee (p \Rightarrow \sigma_2) \vee (q \Rightarrow \sigma_3)$.

Propriétés algébriques Rappelons que $\overrightarrow{\pi} : \Pi \to \mathbb{B}$ désigne un label sur Π, et $\overrightarrow{p} : \mathcal{P} \to \mathbb{B}$ désigne la valuation des paramètres $\mathcal{P}$. Le théorème suivant définit les opérations algébriques attendues, dont la cohérence avec les définitions présentées dans l'introduction peut être vérifiée.

Théorème 1 *Soit Acc_1 et Acc_2 deux ensembles d'acceptation (AS) tels que $Acc_i = (\Pi_i, \mathcal{P}_i, \psi_i, \phi_i)$. Les opérations suivantes définissent une algèbre pour la représentation implicite des AS :*

- *satisfaction : $\alpha \models Acc$ si, et seulement si, $\exists \overrightarrow{p}, [\psi(\overrightarrow{p}) \wedge \forall \overrightarrow{\pi}, \phi(\overrightarrow{p}, \overrightarrow{\pi}) \Leftrightarrow \alpha(\overrightarrow{\pi})]$.*
- *raffinement : $Acc_1 \preceq Acc_2$ si, et seulement si,*
  1. *$\Pi_1 \supseteq \Pi_2$,*
  2. *$\forall \overrightarrow{p_1}, \psi_1(\overrightarrow{p_1}) \Rightarrow [\exists \overrightarrow{p_2}, (\psi_2(\overrightarrow{p_2}) \ \wedge \ \forall \overrightarrow{\pi_1}, (\phi_1(\overrightarrow{p_1}, \overrightarrow{\pi_1}) \Rightarrow \phi_2(\overrightarrow{p_2}, \overrightarrow{\pi_1}\downarrow_{\mathcal{P}_2})))]$.*
- *conjonction : $Acc_1 \wedge Acc_2 = (\Pi, \mathcal{P}, \psi, \phi)$ tel que :*
  - *$\Pi = \Pi_1 \cup \Pi_2$,*
  - *$\mathcal{P} = \mathcal{P}_1 \times \mathcal{P}_2 \times Q$, avec $Q = 2^\Sigma$ et $\Sigma = 2^\Pi$,*
  - *$\psi(p_1, p_2, q) = \psi_1(p_1) \ \wedge \ \psi_2(p_2) \ \wedge \ \Delta(p_1, p_2) \ \wedge \ \Gamma_{p_1,p_2}(q)$, avec :*
    - *$\Delta(p_1, p_2) = \forall \sigma \in \Sigma, (\phi_1(p_1, \sigma) \Leftrightarrow \phi_2(p_2, \sigma))$,*
    - *$\Gamma_{p_1,p_2}(q) = [\forall \sigma \in q, \phi_1(p_1, \sigma_{\downarrow_{\Sigma_1}}) \ \wedge \ \phi_2(p_2, \sigma_{\downarrow_{\Sigma_2}})] \wedge$*
      $[\forall \sigma_1 \in \Sigma_1, \phi_1(p_1, \sigma_1) \Rightarrow \exists \sigma \in q, \sigma_{\downarrow_{\Sigma_1}} = \sigma_1] \wedge$
      $[\forall \sigma_2 \in \Sigma_2, \phi_2(p_2, \sigma_2) \Rightarrow \exists \sigma \in q, \sigma_{\downarrow_{\Sigma_2}} = \sigma_2]$,
  - *$\phi(p_1, p_2, q)(\sigma) = (\sigma \in q)$.*
- *produit : $Acc_1 \otimes Acc_2 = (\Pi, \mathcal{P}, \psi, \phi)$ tel que :*
  - *$\Pi = \Pi_1 \cup \Pi_2$,*
  - *$\mathcal{P} = \mathcal{P}_1 \times \mathcal{P}_2$,*
  - *$\psi(p_1, p_2) = \psi_1(p_1) \ \wedge \ \psi_2(p_2)$,*

  - $\phi(p_1, p_2)(\sigma) = \phi_1(p_1, \sigma_{\downarrow_{\Pi_1}}) \ \wedge \ \phi_2(p_2, \sigma_{\downarrow_{\Pi_2}})$.
- *quotient :* $Acc_1/Acc_2 = (\Pi, \mathcal{P}, \psi, \phi)$ *tel que :*
  - $\Pi = \Pi_1 \cup \Pi_2$,
  - $\mathcal{P} = \mathcal{P}_1 \times Q$, *avec* $Q = 2^{\Sigma}$ *et* $\Sigma = 2^{\Pi}$,
  - $\psi(p_1, p_2) = \psi_1(p_1) \ \wedge \ \forall \sigma \in q, \neg\Delta_{P_2}(q)$, *avec :*
    - $\Delta_{P_2}(q) = \exists p_2 \in \mathcal{P}_2, (\psi_2(p_2) \wedge \phi_2(p_2)(\sigma))$,
  - $\phi(p_1, q)(\sigma) = \phi_1(p_1)(\sigma_{\downarrow_{\Sigma_1}}) \ \wedge \ \Gamma(q)(\sigma)$, *avec* $\Gamma(q)(\sigma) = q \in \sigma$.

Preuve 3 *Nous pouvons vérifier pour chacune de ces opérations que les ensembles calculés sont cohérents avec les définitions du précédent chapitre.*

Pour la conjonction et le quotient, l'ensemble Q est introduit de façon paresseuse. Le principe est de calculer la synchronisation des labels pour construire le graphe des parties pour les labels sur le nouvel alphabet. Nous en déduisons le graphe partiel pour les ready-sets. Dans le pire des cas, l'ensemble Q peut en effet être très grand ($2^{\mathcal{P}}$). En pratique, la taille de cet ensemble est beaucoup plus raisonnable.

Cette représentation réduit la taille des ensembles définis. Considérons l'exemple présenté au début de section, avec les ensembles $A = \{\{a\}\}$ et $B = \{\{b\}\}$. Nous avons vu que la représentation classique de leur quotient aboutissait à un ensemble dont la dimension est très importante. Il peut être défini implicitement avec seulement deux paramètres p et q, et les fonctions caractéristiques $\psi = p \vee \neg q$ et $\phi = (p \Rightarrow \{ab\}) \wedge (q \Rightarrow \{\overline{a}b\})$.

Les représentations implicites des AS fournissent une construction concise. Les fonctions caractéristiques peuvent être codées comme des fonctions booléennes. L'intégration d'une implémentation efficace peut ainsi s'envisager positivement, en s'appuyant par exemple sur des diagrammes de décision binaire (BDD [3] [98]).

3. BDD : Binary Decision Diagram. Plusieurs variantes de BDD existent (RQBDD, ZDD, etc), permettant d'envisager des représentations plus ou moins concises.

Chapitre 7

Automates moniteurs

Les automates moniteurs [9, 10] définissent des systèmes de transitions d'état où chaque état est associé à une formule propositionnelle, et où certains états sont signalés comme illégaux. Tandis qu'un modèle de Kripke représente les comportements possibles d'un système donné, les moniteurs autorisent des observations variées, et permettent ainsi de modéliser la variabilité des comportements envisagés. Les deux formalismes peuvent être liés par les exécutions du modèle, observées avec le moniteur.

Nous proposons de construire une théorie de spécification, autour de ce formalisme, en suivant les principes exposés dans le chapitre 5. La principale caractéristique de cette théorie est que les propriétés des événements et des états peuvent être prises en compte. Cela contraste fortement avec d'autres théories de spécification telles que les automates d'interfaces [3] ou les interfaces modales [13], où les propriétés d'état ne peuvent pas être exprimées nativement. Des exemples typiques de spécification avec des propriétés d'état sont les exigences basées sur des règles où les actions sont rendues obligatoires ou interdites lorsque le système est dans un état donné.

Les moniteurs combinent donc des propriétés d'état, exprimées sous forme de formules propositionnelles, et des propriétés temporelles, exprimées sous forme de séquences d'actions obligatoires ou interdites. Des opérateurs de composition sont fournis, ce qui en fait une théorie complète, avec une relation de raffinement, et des opérateurs pour le produit, la conjonction et le quotient. En substance, les moniteurs étendent deux théories d'interface : les interfaces modales et les interfaces synchrones. Grâce à cette riche algèbre de composition, et au fait que la plupart des exigences comportementales exprimées en langage naturel peuvent être traduites avec les moniteurs, la théorie soutient des méthodes de raisonnement compositionnel, et peut être utilisée pour analyser les exigences comportementales dès le début du processus de conception.

7.1 Définition

Automate moniteur. Un moniteur [9, 10] est un système de transitions d'états où chaque état est associé à une formule propositionnelle, et où certains états sont signalés comme illégaux. Les formules propositionnelles sont étendues par l'ajout de l'opérateur diamant, dont l'interprétation est effectuée en respectant la condition de cohérence de la définition 30.

Définition 28 *Automate moniteur*
Un automate moniteur sur l'alphabet Π *est un tuple* $\mathcal{O} = (\Pi, R, i, \Rightarrow, \mathcal{I}, \varphi)$ *tel que :*

- Π *est un ensemble de propositions atomiques ;*
- R *est un ensemble d'états ;*
- $i \notin R$ *est l'état initial ;*
- $\Rightarrow \subseteq (R \uplus \{i\}) \times R$ *est la relation de transition entre les états ;*
- $\mathcal{I} \subseteq R$ *est l'ensemble des états illégaux ;*
- $\varphi : R \rightarrow \boldsymbol{Prop}(\Pi)$ *est une fonction associant une formule logique à chaque état.*

Invariance par bégaiement. Comme pour les modèles, nous supposons que les moniteurs vérifient la propriété d'invariance par bégaiement.

Définition 29 *Invariance du bégaiement (Stuttering Invariant)*
Les moniteurs sont supposés être invariants par bégaiement, c'est-à-dire $\forall r \in R, r \Rightarrow r$.

Les boucles sont implicites. Elles n'apparaissent pas nécessairement dans les définitions et les représentations. L'ensemble des transitions ($\Rightarrow$) peut facilement être étendu pour inclure ces boucles. Notons également que l'état initial n'est pas invariant par bégaiement, car il n'appartient pas à R.

Exemple 29 *Reprenons l'exemple simplifié sur le comportement d'un ascenseur, avec trois situations envisagées : l'ascenseur est en mouvement (proposition* m*), l'ascenseur est arrêté à l'étage i (propositions* s_0*,* s_1*,* s_2*,* s_3*), la porte de l'ascenseur est ouverte à l'étage i (propositions* o_0*,* o_1*,* o_2*,* o_3*). Nous pouvons représenter toutes les propriétés avec l'alphabet* $\Pi = \{m, s_0, .., s_3, o_0, .., o_3\}$*. La figure 7.1 traduit la règle suivante :*

- *(*R_1*) L'ascenseur doit être mis en marche pour passer d'un étage à l'autre.*

Dans cette figure, les indices i et j sont considérés comme strictement différents ($i \neq j$). L'état illégal r_3 interdit le changement d'état (passage de s_i à s_j) si l'ascenseur n'est pas mis en mouvement ($\neg m$).

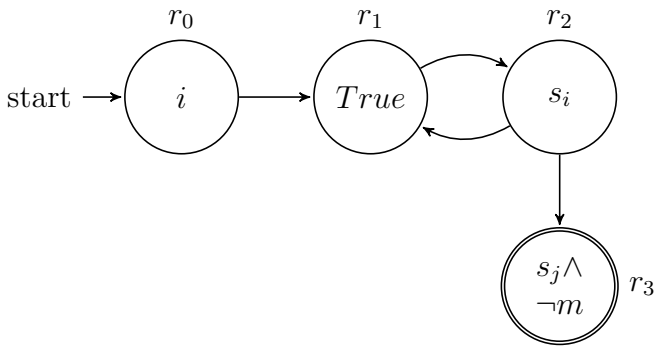


FIGURE 7.1 – L'ascenseur doit être mis en marche pour passer d'un étage à l'autre.

Modalités. Les modalités de la logique modale classique sont directement traduisibles dans les automates moniteurs. L'impossibilité est exprimée directement par les états illégaux. La possibilité est ce qui n'est pas illégal. La nécessité est une proposition dont la négation est interdite : une proposition nécessaire p est spécifiée en rendant $\neg p$ illégal. La contingence est implicite (si la négation d'une proposition n'est pas illégale). La modalité $\diamond(p)$ est introduite pour exprimer le "nécessairement possible", à savoir le fait qu'une proposition p doit être vérifiée au moins dans un des états successeurs de l'état actuel observé.

Définition 30 *Condition de cohérence pour $\diamond(p)$*
La condition de cohérence pour $\diamond(p)$ est vérifiée si, et seulement si, pour tous les états q tels que $\diamond(p) \in \upsilon(q)$, il existe au moins un état q' tel que $q \rightarrow q' \wedge p \in \upsilon(q')$.

Conventions de modélisation. L'expressivité des moniteurs s'étend facilement, en pratique, grâce à l'utilisation de la convention de modélisation. Nous pouvons différencier les éléments des alphabets utilisés pour mettre en évidence les événements. Certaines propositions logiques peuvent ainsi indiquer la réalisation d'un événement. De fait, nous pouvons considérer deux alphabets inclus dans l'alphabet principal, l'un sur les propriétés des états, l'autre sur les événements.

7.2 Configuration, observation et langage support

Un modèle définit un système. Un moniteur définit un ensemble de comportements, c'est-à-dire un ensemble d'exécutions du modèle. Les dynamiques d'observation sont modélisées par les configurations des moniteurs, qui correspondent à des ensembles d'états.

Définition 31 *Configuration*
Soit $\mathcal{O} = (\Pi_O, R, i, \Rightarrow, \mathcal{I}, \varphi)$ *un moniteur .*
Une configuration pour le moniteur $\mathcal{O}$ *est un ensemble d'états* $c \in 2^R \cup \{\{\{i\}\}$.
La configuration $\{i\}$ *est la configuration initiale. L'ensemble des configurations est désigné par* C*. Une configuration est dite illégale si elle contient un état illégal, c'est-à-dire* $c \cap \mathcal{I} \neq \emptyset$ $(\exists r \in c, r \in \mathcal{I})$.

Exemple 30 *La figure 7.2 met en évidence la configuration* $\{r_1, r_2\}$ *pour un moniteur. Cette configuration correspond à la valuation* $\sigma \models True \wedge \neg p$.

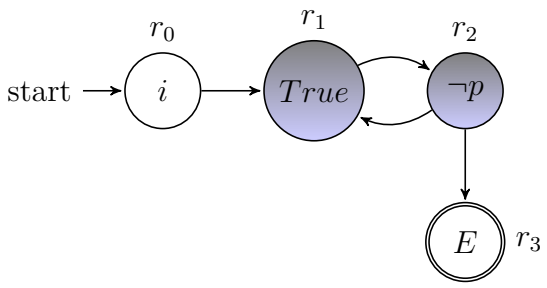


FIGURE 7.2 – Configuration $\{r_1, r_2\}$

Nous introduisons maintenant la notion d'observation, pour formaliser la dynamique d'observation des modèles de réalisation. Plus précisément, nous définissons une fonction d'observation pour relier les séquences de configuration et les états des modèles. Le résultat d'une observation est donc une configuration obtenue à partir d'une configuration donnée et d'un état de modèle (dont on connaît la valuation).

Définition 32 *Fonction d'observation*
Soit $\mathcal{O} = (\Pi_O, R, i, \Rightarrow, \mathcal{I}, \varphi)$ *un moniteur, et* C *l'ensemble des configurations possible sur* $\mathcal{O}$.
La fonction d'observation $obs : C \times 2^{\Pi} \rightarrow C$ *associée avec* $\mathcal{O}$ *définit les configurations observées par* $\mathcal{O}$.

Pour une valuation σ et une configuration c, l'observation de σ en partant de c, notée $obs(c, \sigma)$, est le plus petit ensemble tel que :

1. $\forall r \in c, \forall r' \in R, r \Rightarrow r' \wedge \sigma \models \varphi(r') \Longrightarrow r' \in obs(c, \sigma)$,
2. $\forall r \in obs(c, \sigma), \forall r' \in R, r \Rightarrow r' \wedge \sigma \models \varphi(r') \Longrightarrow r' \in obs(c, \sigma)$.

Une observation initiale est une observation partant de $\{i\}$.

Exemple 31 *La figure 7.3 montre le passage d'une configuration donnée c, à gauche, où l'ascenseur est arrêté à un étage, à une nouvelle configuration c', à droite, où l'ascenseur est en mouvement. La configuration c' est obtenue en calculant l'observation à partir de c d'un ascenseur en mouvement : $c' = obs(c, (m))$. Dans cette figure, les indices i et j sont considérés comme inégaux ($i \neq j$).*

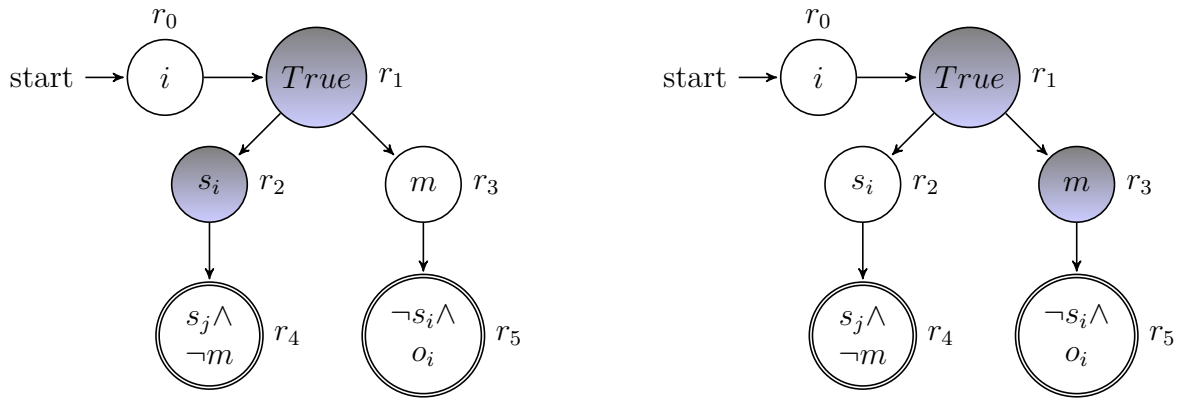


FIGURE 7.3 – L'observation de droite est obtenue à partir de la configuration de gauche avec la valuation (m)

À partir d'une configuration donnée c, il est possible de faire correspondre toutes les séquences u d'un modèle de réalisation avec une observation.

Définition 33 *Observation d'un mot*
Soit c une configuration et u un mot sur 2^{Π}. L'observation $obs(c, u)$ est le plus petit ensemble tel que $obs(c, \sigma.u) = obs(obs(c, \sigma), u)$.

La définition 33 permet finalement de définir le langage support d'un moniteur.

Définition 34 *Langage support d'un automate moniteur*
Le langage support d'un moniteur $\mathcal{O}$ est $\mathcal{L}(\mathcal{O}) = \{u | obs(i, u) \cap C^I = \emptyset\}$.

7.3 Relation de satisfaction

La relation de satisfaction entre les modèles et les moniteurs formalise la conformité d'un modèle à un moniteur. En d'autres termes, un moniteur permettant d'exprimer certaines exigences à respecter, cette relation nous permet de savoir si le modèle respecte ces exigences, ou au contraire les transgresse. En utilisant le concept d'observation, la violation d'une règle se traduit par la mise en évidence d'une observation illégale.

Définition 35 *Relation de satisfaction*
Soit $\mathcal{O}$ un moniteur sur Π_O et $\mathcal{M}$ un modèle sur Π_M, avec $\Pi_O \subseteq \Pi_M$.
$\mathcal{M}$ est un modèle de $\mathcal{O}$, noté $\mathcal{M} \models \mathcal{O}$, si, et seulement si, $\mathcal{L}(\mathcal{M}) \subseteq \mathcal{L}(\mathcal{O})$.

Définition 36 *Simulation*
Soit $\mathcal{O} = (\Pi_O, R, i, \Rightarrow, \mathcal{I}, \varphi)$ un moniteur, et $\mathcal{M} = (\Pi_M, Q, Q_0, \rightarrow, \upsilon)$ un modèle avec $\Pi_O \subseteq \Pi_M$.
La simulation $\varrho \subseteq Q \times R$ est le plus petit ensemble tel que :

1. $\forall q \in Q_0, \forall r \in R,\ i \rightarrow r \wedge \upsilon(q) \models \varphi(r) \Longrightarrow (q, r) \in \varrho$;
2. $\forall (q, r) \in \varrho,\ \forall q' \in Q, \forall r' \in R,\ q \rightarrow q' \wedge r \Rightarrow r' \wedge \upsilon(q') \models \varphi(r') \Longrightarrow (q', r') \in \varrho$.

Remarque 5 *Existence et unicité d'une simulation*
Soit $\mathcal{O}$ un moniteur sur Π_O et $\mathcal{M}$ un modèle sur Π_M, avec $\Pi_O \subseteq \Pi_M$.
Les conditions 1 et 2 de la définition 36 donnent une construction inductive de ϱ. Par conséquent, pour tout moniteur $\mathcal{O}$ sur Π_O and tout modèle $\mathcal{M}$ sur Π_M, avec $\Pi_O \subseteq \Pi_M$, il existe une unique simulation ϱ de $\mathcal{M}$ vers $\mathcal{O}$.

Définition 37 *Simulation consistante*
Soit $\mathcal{O} = (\Pi_O, R, i, \Rightarrow, \mathcal{I}, \varphi)$ un moniteur, et $\mathcal{M} = (\Pi_M, Q, Q_0, \rightarrow, \upsilon)$ un modèle avec $\Pi_O \subseteq \Pi_M$.
La relation $\varrho \subseteq Q \times R$ est dite consistante si l'axiome suivant est vérifié :

— $\forall (q, r) \in \varrho, r \notin \mathcal{I}$ *(condition de consistance).*

Le théorème 2 permet d'avoir une procédure pour décider si un modèle satisfait, ou non, un moniteur sans avoir à déterminer le langage support associé. Cette procédure consiste à calculer, avec une complexité polynomiale, la relation de simulation donnée par la définition 36.

Théorème 2 *Théorème de décision pour la satisfaction*
Soit $\mathcal{O}$ un moniteur sur Π_O et $\mathcal{M}$ un modèle sur Π_M, avec $\Pi_O \subseteq \Pi_M$.
$\mathcal{M}$ est un modèle de $\mathcal{O}$ si, et seulement si, la relation de simulation ϱ de $\mathcal{M}$ par $\mathcal{O}$ est consistante.

Preuve 4

Soit $\mathcal{O}$ un moniteur sur Π_O et $\mathcal{M}$ un modèle sur Π_M, avec $\Pi_O \subseteq \Pi_M$. Nous allons vérifier les deux implications en raisonnant par l'absurde.

Implication 1 : $\mathcal{M} \models \mathcal{O} \Longrightarrow \varrho$ est consistante.

Supposons par l'absurde que ϱ soit inconsistant. Il existe $(q, r) \in \varrho$ avec $r \in \mathcal{I}$. Comme ϱ est inductif, il existe $(q_0, r_0), ..., (q_i, r_i), ..., (q_n, r_n)$, avec $(q_n, r_n) = (q, r)$, de sorte que :

- *$q_0 \in Q_0$ et $i \Rightarrow r_0$ (selon l'axiome 1 de la définition 36) ;*
- *$(q_i, r_i) \in \varrho$, $q_i \rightarrow q_{i+1}$, $r_i \Rightarrow r_{i+1}$, $\upsilon(q) \models \varphi(r_i)$ (selon l'axiome 2 de la définition 36).*

Nous pouvons donc dériver le mot $u = \upsilon(q_0), ..., \upsilon(q_i), ..., \upsilon(q_n)$. Ce mot appartient à $\mathcal{L}(\mathcal{M})$. On peut aussi appliquer la fonction d'observation à u, $obs(\{i\}, u)$, et on a $r \in obs(\{i\}, u)$. Nous déduisons $u \notin \mathcal{L}(\mathcal{O})$, et donc, $\mathcal{M} \not\models \mathcal{O}$. par contraposition, nous vérifions que $\mathcal{M} \models \mathcal{O}$ implique ϱ est consistant.

Implication 2 : ϱ est consistante $\Longrightarrow \mathcal{M} \models \mathcal{O}$.

Supposons par l'absurde que $\mathcal{M} \not\models \mathcal{O}$. Dans ce cas, il existe un mot u tel que $u \in \mathcal{L}(\mathcal{M}) \backslash \mathcal{L}(\mathcal{O})$. On en déduit qu'il existe une séquence $q_0, ..., q_i, ..., q_n$ telle que $q_0 \in Q_0$, $q_i \rightarrow q_{i+1}$, et $obs(\{i\}, \upsilon(q_0)...\upsilon(q_n)) \cap \mathcal{I} \neq \emptyset$. Les moniteurs sont invariants par bégaiement, nous devons donc considérer un écart possible entre les indices de q et r. Par conséquent, nous introduisons l'indice x_i :

- *$\exists m \in \mathbb{N}, \exists (x_i)_{i=0,...,m}$ tel que $x_0 = 0$, $x_{i+1} >= x_i$ et $x_m = n$.*

Finalement, à partir des définitions de la fonction d'observation (définition 32) et de la relation de simulation (définition 36), on déduit qu'il existe $(r_i)_{i=0,...,m}$ tel que $(q_i, r_i) \in \varrho$, $\{i\} \Rightarrow r_0$ et $r_m \in \mathcal{I}$, et donc, ϱ est incohérent. par contraposition, nous vérifions que ϱ est consistant implique que $\mathcal{M} \models \mathcal{O}$.

Remarque 6 *Complexité*
La complexité temporelle du calcul de la simulation ϱ dépend du nombre d'états et de transitions pour un modèle donné et un moniteur donné. Soient n et m les nombres d'états et de transitions du modèle, $N = n+m$, s et t les nombres d'états et de transitions du moniteur et $S = s + t$. La complexité dans le pire des cas est $O(N \times S)$. En pratique, les représentations avec un moniteur sont très denses par rapport aux représentations avec les modèles ; le nombre d'états et de transitions des moniteurs sera généralement beaucoup plus faible que le nombre d'états et de transitions des modèles, et peut être limité par une constante maximale. Ainsi, nous pouvons évaluer la complexité en nous basant uniquement sur les états et les transitions du modèle, c'est-à-dire N. Dans ce contexte, la complexité pour le calcul de la simulation ϱ est $O(N)$, c'est-à-dire linéaire.

7.4 Quelques opérations de transformation

Cette section présente quelques opérations utiles. La réduction permet, sans perte d'expressivité, de réduire le nombre d'états d'un moniteur. *A contrario*, la complétion permet d'enrichir un moniteur pour assurer l'observation explicite de tout comportement de modèles. Nous introduisons finalement la notion de moniteur déterministe, utile pour définir le raffinement.

7.4.1 Réduction

Selon la définition de la relation de satisfaction, les comportements non observés sont validés car ils ne permettent pas d'accéder à un état illégal. Les états qui ne permettent pas l'accès à un état illégal n'apportent donc aucun gain en termes d'expressivité, et nous pouvons construire un moniteur équivalent réduit en supprimant ces états.

Définition 38 *Moniteur réduit*
Un moniteur $\mathcal{O}$ est dit réduit si, et seulement si, la condition suivante est vérifiée :

(i) $\forall r \in R$, il existe un chemin $r, r_1, ..., r_m$ tel que $r \Rightarrow r_1$, $\forall i, r_i \Rightarrow r_{i+1} \wedge \varphi(r_i)$ peut être vérifiée et $r_m \in \mathcal{I}$.

Exemple 32 *La figure 7.4 montre deux moniteurs, $\mathcal{O}_1$ et $\mathcal{O}_2$. Le moniteur $\mathcal{O}_2$ (à droite) est obtenu en appliquant une réduction sur le moniteur $\mathcal{O}_1$ (à gauche). Les états qui n'autorisent pas l'accès à un état illégal sont supprimés.*

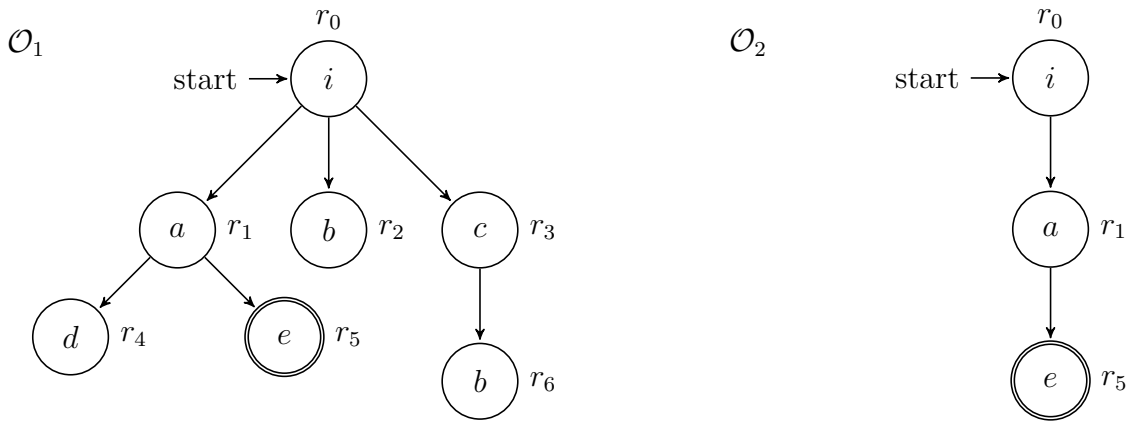


FIGURE 7.4 – Un moniteur (à gauche) et sa réduction (à droite)

7.4.2 Complétion

Un moniteur est une structure qui permet d'observer le comportement d'un modèle. Il est toutefois possible que certains comportements ne soient pas observables par un moniteur donné. Selon la définition de la relation de satisfaction (définition 35), les comportements non observés sont valides (c'est-à-dire non illégaux). Cependant, il peut être utile d'observer explicitement tout comportement. Nous proposons ici une opération de transformation pour construire, à partir d'un moniteur donné $\mathcal{O}$, un moniteur équivalent $\mathcal{O}'$ qui observe explicitement tout comportement de tout modèle. Ce moniteur est dit complet.

Définition 39 *Moniteur complet*
Un moniteur $\mathcal{O}$ est dit complet si, et seulement si, la condition suivante est vérifiée :

(i) $\forall \sigma \in 2^{\Pi}, \forall r \in R, \exists r' \in R, r \Rightarrow r' \wedge \upsilon(\sigma) \models \varphi(r')$.

Exemple 33 *La figure 7.5 montre deux moniteurs, $\mathcal{O}_1$ et $\mathcal{O}_2$. Le moniteur $\mathcal{O}_2$ (à droite) est obtenu en appliquant une complétion sur le moniteur $\mathcal{O}_1$ (à gauche).*

Remarque 7 *Il existe de nombreuses façons de compléter un moniteur. L'exemple 33 montre une solution préservant le déterminisme d'un moniteur (s'il est initialement déterministe). Il est également possible d'ajouter simplement un état associé à la formule Vrai (true), et de relier chaque état à ce nouvel état (cette solution ne préserve pas le déterminisme).*

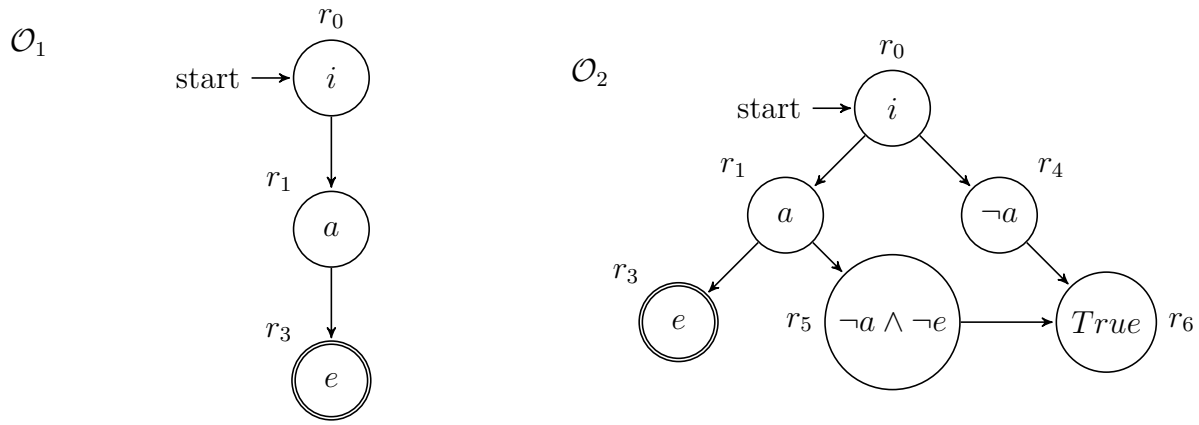


FIGURE 7.5 – Un moniteur (à gauche) et sa complétion (à droite)

7.4.3 Déterminisation

Nous introduisons la notion de déterminisation pour un moniteur, utile pour définir le raffinement du moniteur.

Définition 40 *Moniteur déterministe*

Un moniteur $\mathcal{O}$ est dit déterministe si, et seulement si, la condition suivante est vérifiée :

(i) $\forall r \in R \cup \{i\}, \forall r_1, r_2 \in R, r \Rightarrow r_1$, $r \Rightarrow r_2$ *et* $r_1 \neq r_2$ *implique que* $\varphi(r_1) \wedge \varphi(r_2)$ *ne peut pas être satisfait.*

Exemple 34 *La figure 7.6 montre deux moniteurs, $\mathcal{O}_1$ et $\mathcal{O}_2$. Le moniteur $\mathcal{O}_2$ (à droite) est obtenu en appliquant une opération de déterminisation sur le moniteur $\mathcal{O}_1$ (à gauche).*

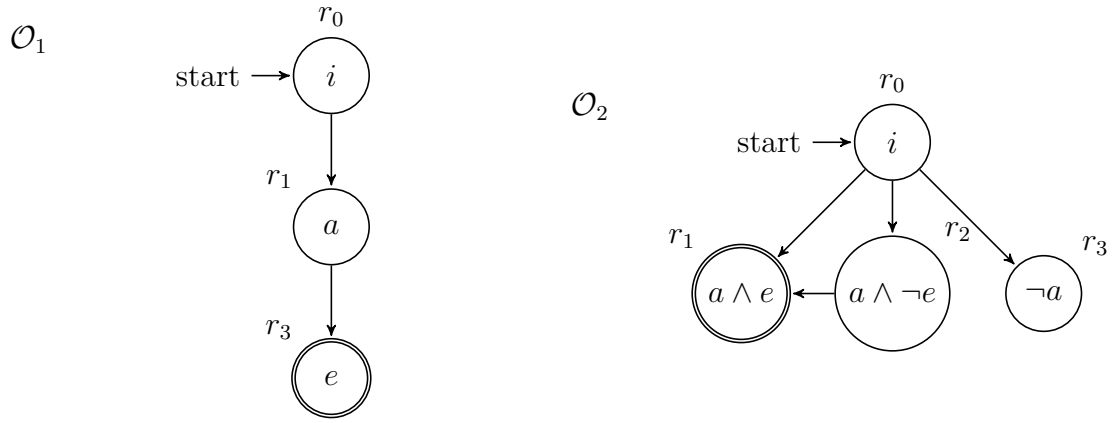


FIGURE 7.6 – Un automate moniteur (à gauche) et sa déterminisation (à droite)

Corollaire 1

Soit $\mathcal{O}$ un moniteur sur Π_O.
La propriété suivante est vérifiée si $\mathcal{O}$ est déterministe :

(i) $\forall r, r' \in R, r \neq r'$ et $r \Rightarrow r'$ implique $\varphi(r) \wedge \varphi(r')$ ne peut pas être satisfait.

Avant de proposer un algorithme pour construire un moniteur déterministe à partir d'un moniteur donné, nous introduisons quelques fonctions utiles.

Tout d'abord, nous définissons la fonction successeur qui permet d'obtenir une nouvelle configuration à partir d'une configuration et d'une formule données. La nouvelle configuration correspond à l'observation à partir de la configuration initiale et à une valuation qui satisfait la formule donnée :

— $successor : Monitor \times Config \times \mathbf{Prop}(\Pi) \rightarrow Config$
$successor(\mathcal{O}, c, \varphi)$ sélectionne une valuation $\sigma \in 2^{\Pi}$ telle que $\sigma \models \varphi$ φ peut être satisfaite et retourne la configuration $c' = obs(c, \sigma)$ (retourne un ensemble vide si φ ne peut pas être satisfaite).

Nous proposons ensuite quelques fonctions pour définir la formule qui sera associée à la configuration obtenue :

— $pre\_adherence : Monitor \times Config \times Config \rightarrow Config$
$pre\_adherence(\mathcal{O}, c, c')$ retourne l'ensemble $\{r | r \in c \backslash c'$ et $\exists r' \in c', r \Rightarrow r'\}$

— $post\_adherence : Monitor \times Config \rightarrow Config$
$post\_adherence(\mathcal{O}, c)$ retourne l'ensemble $\{r' | r' \notin c$ et $\exists r \in c, r \Rightarrow r'\}$

— $support\_formula : Monitor \times Config \times Config \rightarrow \mathbf{Prop}(\Pi)$
$support\_formula(\mathcal{O}, c, c')$ retourne la formule suivante :

$$\varphi' = \bigwedge_{x \in pre} \neg\varphi(x) \wedge \bigwedge_{x \in c'} \varphi(x) \wedge \bigwedge_{x \in post} \neg\varphi(x)$$

avec $pre = pre\_adherence(\mathcal{O}, c, c')$ et $post = post\_adherence(\mathcal{O}, c')$

Enfin, nous définissons une fonction pour dériver une nouvelle paire associant configuration et formule à partir d'une configuration et d'une formule données.

— $derivation : Monitor \times Config \times \mathbf{Prop}(\Pi) \rightarrow (Config, \mathbf{Prop}(\Pi))$
$derivation(\mathcal{O}, c, \varphi)$ retourne une paire (c', φ') telle que $c' = successor(\mathcal{O}, c, \varphi)$ et $\varphi' = support\_formula(\mathcal{O}, c, c')$

Exemple 35 *La figure 7.7 montre deux configurations d'un moniteur donné $\mathcal{O}$. La configuration c' est le résultat de successor$(\mathcal{O}, c, \varphi)$, avec $\varphi = \neg a$. $\sigma = (bc)$ est une valuation possible telle que $\sigma \models \varphi$. L'application des différentes fonctions donne les résultats suivants :*

- $successor(\mathcal{O}, c, \varphi) = c'$
- $pre\_adherence(\mathcal{O}, c, c') = \{r_1\}$
- $post\_adherence(\mathcal{O}, c, c') = \{r_1, r_4\}$
- $support\_formula(\mathcal{O}, c, c') = (\neg a) \wedge (True \wedge b \wedge c) \wedge (\neg a \wedge \neg d) = \neg a \wedge b \wedge c \wedge \neg d$
- $derivation(\mathcal{O}, c, \varphi) = (c', \neg a \wedge b \wedge c \wedge \neg d)$

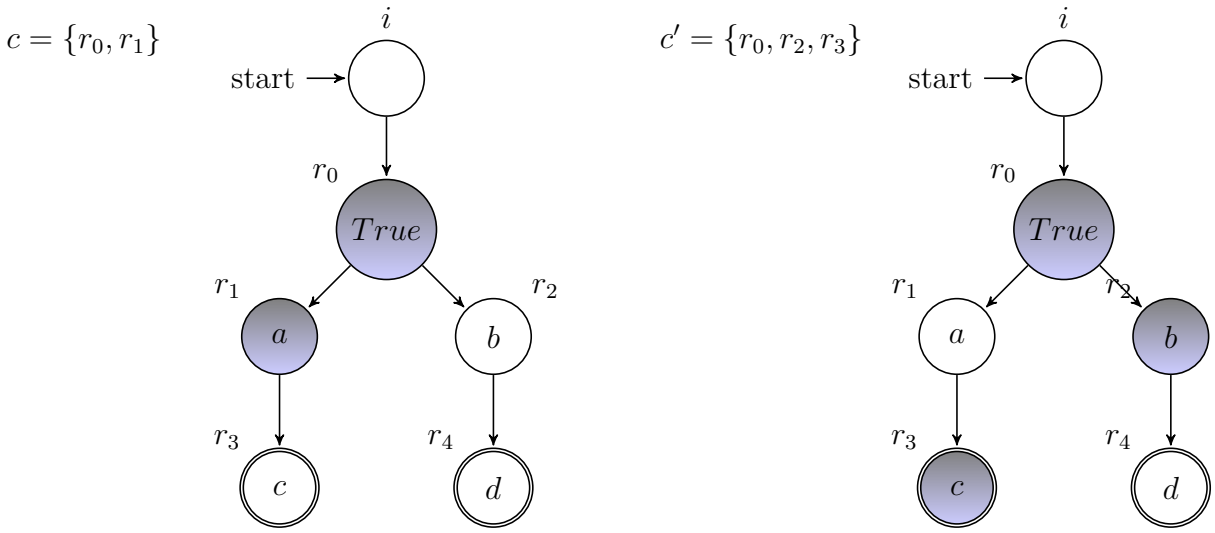


FIGURE 7.7 – Résultat du successeur pour $\varphi = \neg a$ ((bc) est une valuation possible)

Enfin, l'algorithme 2 est utilisé pour produire un moniteur déterministe noté $\mathcal{O}^{det}$. Chaque état du moniteur déterministe est relié à une paire associant une configuration et une formule. Il existe deux fonctions principales, la déterminisation en largeur ($DetL$) et la déterminisation en profondeur ($DetP$), mutuellement récursives. La fonction $DetP$ est appelée en premier, à partir de l'état initial (correspondant à la configuration initiale). Elle construit en profondeur les automates déterministes, chaque état correspondant à une paire (configuration, formule). La fonction $DetL$ est appelée pour ajouter tous les successeurs possibles des états ajoutés. Si l'un des successeurs est un nouvel état, la fonction $DetL$ est appelée à nouveau pour cet état, et ainsi de suite jusqu'au point fixe.

Algorithm 2 DÉTERMINISATION

Entrée : $\mathcal{O} = (\Pi, R, i, \rightarrow, \varphi, \mathcal{I})$
Sortie : $\mathcal{O}^{det} = (\Pi, R^{det}, i, \Rightarrow^{det}, \varphi^{det}, \mathcal{I}^{det})$
Ensure: $\mathcal{O}^{det}$ est déterministe

function DETL($\mathcal{O}$, $\mathcal{O}^{det}$, Z, $(r,(c,\varphi))$, α)
  if $\alpha \neq False$ **then** ▷ α peut être satisfait
    $c', \varphi' \leftarrow$ DERIVATION($\mathcal{O}$, c, α)
    nouvel état r' correspondant à (c', φ')
    if $(r', (c', \varphi')) \notin Z$ **then**
      $\mathcal{O}^{det}$, $Z \leftarrow$ DETP($\mathcal{O}$, $\mathcal{O}^{det}$, Z, $(r', (c', \varphi'))$)
    end if
    add $r \Rightarrow^{det} r'$
    $\alpha \leftarrow \alpha \wedge \neg\varphi'$ ▷ Mise à jour de la formule d'acceptation du successeur
    return DETL($\mathcal{O}$, $\mathcal{O}^{det}$, Z, $(r,(c,\varphi))$, α)
  else
    return $\mathcal{O}^{det}$
  end if
end function

function DETP($\mathcal{O}$, $\mathcal{O}^{det}$, Z, $(r,(c,\varphi))$)
  $\mathcal{R}^{det} \leftarrow r$ ▷ Ajouter nouvel état à $\mathcal{O}^{det}$
  define $\varphi^{det}(r) = \varphi$
  if c' is illegal **then** $\mathcal{I}^{det} \leftarrow r$
  end if
  $Z \leftarrow (r,(c,\varphi))$ ▷ Mise à jour de l'ensemble des états visités
  $\alpha \leftarrow True$
  $\mathcal{O}^{det}$, $Z \leftarrow$ DETL($\mathcal{O}$, $\mathcal{O}^{det}$, Z, $(r,(c,\varphi))$, α)
  return $\mathcal{O}^{det}$
end function

$\mathcal{O}^{det} := (\Pi, \emptyset, i, \emptyset, \emptyset, \emptyset)$
$Z \leftarrow \emptyset$ ▷ Initialisation de l'ensemble des états visités
$(r,(c,\varphi)) \leftarrow (i, (\{i\}, True))$ ▷ Départ de l'état initial
return DETP($\mathcal{O}$, $\mathcal{O}^{det}$, Z, $(r,(c,\varphi))$)

La validité de l'algorithme est vérifiée en démontrant les trois propriétés suivantes :

(i) l'algorithme termine ;

(ii) le moniteur $\mathcal{O}^{det}$ (résultat) est déterministe ;

(iii) le moniteur $\mathcal{O}^{det}$ (résultat) est équivalent au moniteur $\mathcal{O}$ (source) dans le sens où $\mathcal{L}(\mathcal{O}) = \mathcal{L}(\mathcal{O}^{det})$, c'est-à-dire que tous les modèles de $\mathcal{O}$ sont des modèles de $\mathcal{O}^{det}$ et vice versa.

Nous devons d'abord nous assurer que l'algorithme se termine et que le moniteur qui en résulte est déterministe.

Propriété 3 *L'algorithme 2 termine.*

Preuve 5 *Il existe un nombre fini de configurations possibles (parce que le nombre d'états du moniteur source $\mathcal{O}$ est fini). De même, il existe un nombre fini de formules logiques sur l'alphabet Π (l'alphabet Π étant fini). Chaque état du moniteur résultat $\mathcal{O}^{det}$ est défini comme une paire associant une configuration et une formule logique. Par conséquent, nous avons un nombre fini d'états, et un nombre fini de transitions entre ces états. L'algorithme se termine lorsque chaque état a été traité.*

Propriété 4 *L'application de l'algorithme 2 donne un moniteur déterministe.*

Preuve 6 *Les états initiaux de $\mathcal{O}$ et $\mathcal{O}^{det}$ sont identiques. Nous considérons les successeurs pour chaque état de $\mathcal{O}^{det}$. Selon l'algorithme, à partir d'un état de $\mathcal{O}^{det}$, un successeur n'est ajouté que s'il correspond à l'un des cas suivants : φ_1, $\neg\varphi_1 \wedge \varphi_2$, $\neg\varphi_1 \wedge \neg\varphi_2 \wedge \varphi_3$, etc. Ces formules définissent des ensembles disjoints ; il est donc impossible d'avoir une valuation qui vérifierait deux de ces formules en même temps. Ce point étant vérifié pour chaque état, le moniteur $\mathcal{O}^{det}$ est déterministe.*

Nous voulons maintenant vérifier que le moniteur source $\mathcal{O}$ et le moniteur résultat $\mathcal{O}^{det}$ sont équivalents dans le sens où $\mathcal{L}(\mathcal{O}) = \mathcal{L}(\mathcal{O}^{det})$, c'est-à-dire que tous les modèles de $\mathcal{O}$ sont des modèles de $\mathcal{O}^{det}$ et vice versa. Nous commençons par présenter et démontrer deux lemmes (lemmes 1 et 2).

Lemme 1 *Soit $\mathcal{O}$ un moniteur sur Π, et $\mathcal{O}^{det}$ le moniteur obtenu en appliquant l'algorithme 2 de $\mathcal{O}$. Toute observation de $\mathcal{O}$ peut être associée à un ensemble de chemins dans $\mathcal{O}$, et cet ensemble correspond à un chemin unique dans $\mathcal{O}^{det}$.*

Preuve 7 *Nous considérons tous les mots u définis chacun comme une séquence de valuations. Selon la définition 33, l'observation de tout mot u peut être reliée à une séquence de configurations. Soit $c_0, c_1, ..., c_n$ cette séquence. Considérons deux configurations successives c_i, c_{i+1} de cette séquence. Selon la définition 32, pour chaque état r' de c_{i+1}, il existe un état r tel que $r \Rightarrow r'$ (r prédécesseur de r') et $r \in c_i$, ou $r \in c_{i+1}$. De plus, l'observation est une construction récursive : à partir de c_i, il est possible de calculer un premier ensemble s_1 avec les successeurs des états $\in c_i$ (axiome 1 de la définition 32), puis un ensemble s_2 avec les successeurs des états $\in s_1$ (axiome 2 de la définition 32), et ainsi de suite jusqu'au point fixe. On peut donc relier à cette séquence tous les chemins partant des états de c_i et se terminant dans chaque état de c_{i+1}, et par extension de c_0 à c_n. Cette propriété est vérifiée pour toute observation, et donc pour toute observation partant de l'état initial. L'algorithme 2 donne un moniteur déterministe $\mathcal{O}^{det}$. Par conséquent, chaque état de $\mathcal{O}^{det}$ n'a qu'un seul successeur correspondant à une valuation donnée. De plus, il n'y a qu'un seul chemin correspondant à un mot donné u à partir de l'état initial.*

Lemme 2 *Soit $\mathcal{O}$ un moniteur, et $\mathcal{O}^{det}$ le moniteur obtenu en appliquant l'algorithme 2 de $\mathcal{O}$. On note $S_{\mathcal{O}}$ tous les ensembles de chemins de $\mathcal{O}$, et p^{det} tous les chemins de $\mathcal{O}^{det}$. La propriété suivante est vérifiée pour tous les $S_{\mathcal{O}}$ de sorte que chaque chemin de $S_{\mathcal{O}}$ correspond à un unique p^{det} : s'il y a un chemin $p \in S_{\mathcal{O}}$ qui se termine dans un état illégal, alors p^{det} se termine dans un état illégal.*

Preuve 8 *Tout état de $\mathcal{O}^{det}$ associé à une configuration est illégal si cette configuration est illégale. Selon l'algorithme, les configurations correspondent à une observation. Pour tout mot, selon le lemme 1, on peut associer un ensemble de chemins sur $\mathcal{O}$ à un chemin unique sur $\mathcal{O}^{det}$. Si l'un des chemins se termine par un état illégal, cela signifie que l'un des états de la configuration finale est illégal. Cette configuration est associée à l'état final du chemin sur $\mathcal{O}^{det}$, qui est donc illégal.*

Nous pouvons maintenant compléter la preuve de la validité de l'algorithme.

Propriété 5 *Équivalence entre $\mathcal{O}$ et $\mathcal{O}^{det}$*
Soit $\mathcal{O}$ un moniteur. En appliquant l'algorithme 2 à partir de $\mathcal{O}$, dont le résultat est $\mathcal{O}^{det}$, on vérifie la propriété suivante :

- *le moniteur $\mathcal{O}^{det}$ (résultat) est équivalent à $\mathcal{O}$ (source) dans le sens où $\mathcal{L}(\mathcal{O}) = \mathcal{L}(\mathcal{O}^{det})$, c'est-à-dire que tous les modèles de $\mathcal{O}$ sont des modèles de $\mathcal{O}^{det}$ et vice versa.*

Preuve 9 *Nous considérons les langages $\mathcal{L}(\mathcal{O})$ et $\mathcal{L}(\mathcal{O}^{det})$ comme les langages support respectivement de $\mathcal{O}$ et $\mathcal{O}^{det}$. Selon la définition 34, chaque mot u de $\mathcal{L}(\mathcal{O})$ peut être associé à une observation $obs(i,u)$ telle que $obs(i,u) \cap C^I = \emptyset$. Selon le lemme 2, si $obs(i,u) \cap C^I = \emptyset$, alors le chemin correspondant dans $\mathcal{O}^{det}$ ne se termine pas par un état illégal. Il définit donc un mot de $\mathcal{L}(\mathcal{O}^{det})$. On peut appliquer le même raisonnement aux mots qui n'appartiennent pas à $\mathcal{L}(\mathcal{O})$ (ce qui prouve, par contraposition, que chaque mot de $\mathcal{L}(\mathcal{O}^{det})$ est un mot de $\mathcal{L}(\mathcal{O})$). Nous vérifions donc l'équivalence des deux langages support $\mathcal{L}(\mathcal{O})$ et $\mathcal{L}(\mathcal{O}^{det})$.*

Propriété 6 *Propriété sur la simulation de $\mathcal{M}$ à $\mathcal{O}$*
Soit $\mathcal{O}$ un moniteur sur Π_O, $\mathcal{O}^{det}$ sa déterminisation et $\mathcal{M}$ un modèle sur Π_M, avec $\Pi_O \subseteq \Pi_M$.
La simulation de $\mathcal{M}$ à $\mathcal{O}^d$ est une fonction partielle injective de Q à R.

Preuve 10 *Cette propriété est une conséquence directe de la déterminisation du moniteur.*

7.5 Relation de raffinement

La relation de raffinement entre deux moniteurs formalise l'idée de conformité d'un moniteur par rapport à un autre moniteur. L'intérêt de cette relation est de permettre la comparaison de moniteurs portant sur le même système ou composant, à des niveaux d'abstraction différents.

Définition 41 *Relation de raffinement*
$\mathcal{O}_1$ est un raffinement de $\mathcal{O}_2$, noté $\mathcal{O}_1 \preceq \mathcal{O}_2$, si, et seulement si, pour tous les modèles $\mathcal{M}$, $\mathcal{M} \models \mathcal{O}_1 \Longrightarrow \mathcal{M} \models \mathcal{O}_2$.

Nous définissons maintenant une relation de simulation entre deux moniteurs, utile pour déterminer si un moniteur donné $\mathcal{O}_1$ est un raffinement d'un autre moniteur $\mathcal{O}_2$. Le point important de cette relation implique de considérer le champ d'observation de chaque paire d'états, r_1 de $\mathcal{O}_1$ et r_2 de $\mathcal{O}_2$. Un état r_1 de $\mathcal{O}_1$ est lié à un état r_2 de $\mathcal{O}_2$ si le champ d'observation de r_1 intercepte le champ d'observation de r_2, en ce sens qu'une série de modèles (ou de mots) observée par r_1 serait également observée par r_2.

Définition 42 *Simulation entre deux moniteurs*
Soient $\mathcal{O}_1$ et $\mathcal{O}_2$ deux moniteurs . La relation $\eta \subseteq R_1 \times R_2$ définit une simulation de $\mathcal{O}_1$ par $\mathcal{O}_2$, noté $\eta(\mathcal{O}_1, \mathcal{O}_2)$, si, et seulement si :

(i) $\forall r_1 \in R_1, \forall r_2 \in R_2,$
$(i \Rightarrow_1 r_1) \wedge (i \Rightarrow_2 r_2) \wedge (\varphi(r_1) \wedge \varphi(r_2)$ *satisfiable*$) \Longrightarrow (r_1, r_2) \in \eta$

(ii) $\forall (r_1, r_2) \in \eta,\ \forall r_1' \in R_1, \forall r_2' \in R_2,$
$(r_1 \Rightarrow_1 r_1') \wedge (r_2 \Rightarrow_2 r_2') \wedge (\varphi(r_1') \wedge \varphi(r_2')$ *satisfiable*$) \Longrightarrow (r_1', r_2') \in \eta$

La définition 43 formalise la notion de simulation cohérente, nécessaire pour démontrer le raffinement.

Définition 43 *Simulation consistante entre deux moniteurs*
Soient $\mathcal{O}_1$ et $\mathcal{O}_2$ deux moniteurs . La simulation $\eta(\mathcal{O}_1, \mathcal{O}_2)$ est dite cohérente si l'axiome suivant est respecté :

— $\forall (r_1, r_2) \in \eta, r_1 \notin \mathcal{I} \Longrightarrow r_2 \notin \mathcal{I}$.

La définition 43 est suffisante pour démontrer le raffinement si $\mathcal{O}_1$ est déterministe, c'est-à-dire si chaque observation d'un mot correspond à un chemin unique dans $\mathcal{O}_1$, ce qui permet de déterminer les chemins correspondants dans $\mathcal{O}_2$. En particulier, si un mot est accepté dans $\mathcal{O}_1$, on peut directement vérifier avec cet axiome s'il est systématiquement accepté par $\mathcal{O}_2$, ou non. $\mathcal{O}_1 \preceq \mathcal{O}_2$ si, et seulement si, $\mathcal{L}(\mathcal{O}_1) \subseteq \mathcal{L}(\mathcal{O}_2)$ est déduit en appliquant les définitions 35 et 41. Il est enfin possible de proposer un théorème utilisable pour décider du raffinement entre deux moniteurs, sur la base de la relation de simulation qui est directement calculable.

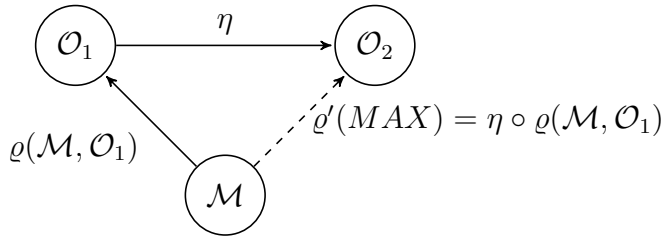
Théorème 3 *Théorème de décision pour le raffinement*
Soient $\mathcal{O}_1$ et $\mathcal{O}_2$ deux moniteurs, avec $\mathcal{O}_1$ déterministe et complet. $\mathcal{O}_1 \preceq \mathcal{O}_2$ si, et seulement si, la simulation $\eta(\mathcal{O}_1, \mathcal{O}_2)$ est consistante (définition 43).

Preuve 11

Soient $\mathcal{O}_1$ et $\mathcal{O}_2$ deux moniteurs, avec $\mathcal{O}_1$ déterministe et complet. Soit $\eta(\mathcal{O}_1, \mathcal{O}_2)$ la simulation de $\mathcal{O}_1$ à $\mathcal{O}_2$ remplissant les conditions de la définition 42. Considérons les relations de simulation $\varrho(\mathcal{M}, \mathcal{O}_1)$ et $\varrho(\mathcal{M}, \mathcal{O}_2)$ entre tout modèle $\mathcal{M}$ et respectivement $\mathcal{O}_1$, $\mathcal{O}_2$.

Implication 1 : $\eta(\mathcal{O}_1, \mathcal{O}_2)$ *consistant* $\Longrightarrow \mathcal{O}_1 \preceq \mathcal{O}_2$

Soit $\mathcal{M}$ *un modèle tel que* $\mathcal{M} \models \mathcal{O}_1$*. La relation de simulation* $\varrho(\mathcal{M}, \mathcal{O}_1)$ *est cohérente (Définition 37). Considérons la relation* $\varrho_2 = \eta(\mathcal{O}_1, \mathcal{O}_2) \circ \varrho(\mathcal{M}, \mathcal{O}_1)$*. Nous avons* $\varrho_2 \subseteq Q \times R_2$ *par construction. Nous allons prouver que* $\varrho(\mathcal{M}, \mathcal{O}_2) \subseteq \varrho_2$ *et que* ϱ_2 *est cohérent. Cela prouverait que* $\varrho(\mathcal{M}, \mathcal{O}_2)$ *est également cohérent, et donc* $\mathcal{M} \models \mathcal{O}_2$*.*



Considérons les ensembles support $\mathcal{R}_n$ *de* $\mathcal{O}_2$ *tels que :*

1. $\mathcal{R}_0 = \{r | i \Rightarrow_2 r\}$,
2. $\mathcal{R}_{n+1} = \{r' | r \Rightarrow_2 r' \wedge r \in \mathcal{R}_n\} \cup \mathcal{R}_n$.

Remarquons que, par construction, $\forall n, \mathcal{R}_n \subseteq R_2$ *et il existe* m *tel que* $\mathcal{R}_m$ *contient tous les états atteignables de* $\mathcal{O}_2$ *(et donc,* $\forall n > m, \mathcal{R}_n = \mathcal{R}_m$*). Donc, considérons la propriété suivante* $P_n : \forall r_2 \in \mathcal{R}_n, (q, r_2) \in \varrho(\mathcal{M}, \mathcal{O}_2) \Longrightarrow (q, r_2) \in \varrho_2$*. Nous prouvons que* P_n *est vrai pour tout* n*, et donc* $\varrho(\mathcal{M}, \mathcal{O}_2) \subseteq \varrho_2$ *:*

1. $\forall q_0 \in Q_0, \forall r_2 \in \mathcal{R}_0$ *tel que* $i \Rightarrow_2 r_2$*, nous avons* $(q_0, r_2) \in \varrho(\mathcal{M}, \mathcal{O}_2)$ *si* $\upsilon(q_0) \models \varphi(r_2)$ *(définition 36, condition 1). Selon l'hypothèse que* $\mathcal{O}_1$ *est complet, il y a nécessairement un état* $r_1 \in R_1$ *tel que* $i \Rightarrow_1 r_1 \wedge \upsilon(q_0) \models \varphi(r_1)$*. D'après la condition 1 de la définition 42, nous avons* $(r_1, r_2) \in \eta(\mathcal{O}_1, \mathcal{O}_2)$ *par construction, et* $(q_0, r_2) \in \varrho_2$ *(*$(r_1, r_2) \circ (q_0, r_1) = (q_0, r_2)$*), ce qui prouve que* P_0 *est vérifiée.*

2. *Supposons que* P_n *est vérifiée :* $\forall r_2 \in \mathcal{R}_n$*, nous avons* $(q, r_2) \in \varrho(\mathcal{M}, \mathcal{O}_2) \Longrightarrow (q, r_2) \in \varrho_2$*. Pour tout* $(q, r_2) \in \varrho(\mathcal{M}, \mathcal{O}_2)$*, tout* $(q, r_2) \in \varrho(\mathcal{M}, \mathcal{O}_2)$*, tout* $q' \in Q$ *et* $r'_2 \in R_2$ *tels que* $r_2 \Rightarrow_2 r'_2$*, nous avons* $(q', r'_2) \in \varrho(\mathcal{M}, \mathcal{O}_2)$ *si* $\upsilon(q') \models \varphi(r'_2)$ *(definition 36, condition 2). D'après l'hypothèse que* $\mathcal{O}_1$ *est complet, il y a nécessairement un état* $r'_1 \in R_1$ *tel que* $r_1 \Rightarrow_1 r'_1 \wedge \upsilon(q') \models \varphi(r'_1)$*. D'après la condition 2 de la définition 42, nous avons* $(r'_1, r'_2) \in \eta(\mathcal{O}_1, \mathcal{O}_2)$ *par construction, et* $(q', r'_2) \in \varrho_2$ *(*$(r'_1, r'_2) \circ (q', r'_1) = (q', r'_2)$*). Nous vérifions que* $\forall r'_2 \in \mathcal{R}_{n+1}, (q', r'_2) \in \varrho(\mathcal{M}, \mathcal{O}_2) \Longrightarrow \varrho_2$*, ce qui prouve donc que* P_{n+1} *est vérifiée.*

Nous concluons donc que P_n vaut pour tout n et que $\varrho(\mathcal{M}, \mathcal{O}_2) \subseteq \varrho_2$.

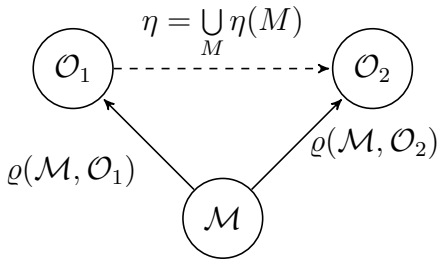
Nous allons maintenant prouver que ϱ_2 est consistant :

— *Nous avons $\forall (r_1, r_2) \in \eta, r_1 \notin \mathcal{I}_1 \Rightarrow r_2 \notin \mathcal{I}_2$ (definition 43). Pour tout $(q, r_1) \in \varrho$, nous avons $r_1 \notin \mathcal{I}_1$ (definition 36, condition 3). Par construction, pour tout $(q, r_2) \in \varrho_2$, il existe r_1 tel que $(r_1, r_2) \in \eta(\mathcal{O}_1, \mathcal{O}_2)$ et $(q, r_1) \in \varrho(\mathcal{M}, \mathcal{O}_1)$. D'après la définition 43, nous déduisons donc que $r_2 \notin \mathcal{I}_2$.*

Nous concluons donc que $\varrho(\mathcal{M}, \mathcal{O}_2)$ est également cohérent, et donc $\mathcal{M} \models \mathcal{O}_2$.

Implication 2 : $\mathcal{O}_1 \preceq \mathcal{O}_2 \Longrightarrow \eta(\mathcal{O}_1, \mathcal{O}_2)$ consistant.

Soit $\mathcal{M}$ un modèle. Nous considérons $\eta(\mathcal{M}) = \varrho(\mathcal{M}, \mathcal{O}_2) \circ \varrho(\mathcal{M}, \mathcal{O}_1)^{-1}$. Rappelons que $\varrho(\mathcal{M}, \mathcal{O}_2) \subseteq Q \times R_2$, $\varrho(\mathcal{M}, \mathcal{O}_1) \subseteq Q \times R_1$. Nous avons donc $\varrho(\mathcal{M}, \mathcal{O}_1)^{-1} \subseteq R_1 \times Q$ et $\eta(\mathcal{M}) \subseteq R_1 \times R_2$. Définir η comme l'union de tous les $\eta(M)$: $\eta = \bigcup_{\mathcal{M}} \eta(\mathcal{M})$. Nous prouverons que $\eta(\mathcal{O}_1, \mathcal{O}_2) \subseteq \eta$ et que η est cohérent. Cela prouverait que $\eta(\mathcal{O}_1, \mathcal{O}_2)$ est également cohérent.



Considérons les ensembles de soutien $\mathcal{R}_{1,n}$ de $\mathcal{O}_1$ et $\mathcal{R}_{2,n}$ de $\mathcal{O}_2$ tels que, pour $i = 1, 2$:

1. $\mathcal{R}_{i,0} = \{r | i \Rightarrow_i r\}$,
2. $\mathcal{R}_{i,n+1} = \{r' | r \Rightarrow_i r' \wedge r \in \mathcal{R}_{i,n}\}\} \cup \mathcal{R}_{i,n}\}$.

On peut remarquer que, par construction, $\forall n, \mathcal{R}_{i,n} \subseteq R_i$ et il existe m tel que $\mathcal{R}_{i,m}$ contient tous les états accessibles de $\mathcal{O}_i$ (et donc, $\forall n > m, \mathcal{R}_{i,n} = \mathcal{R}_{i,m}$). Donc, considérons la propriété suivante P_2 : $\forall r_1 \in \mathcal{R}_{1,n}, r_2 \in \mathcal{R}_{2,n}, (r_1, r_2) \in \varrho(\mathcal{O}_1, \mathcal{O}_2) \Longrightarrow (r_1, r_2) \in \eta$. Nous prouvons que P_n est vrai pour tous les n, et donc $\eta(\mathcal{O}_1, \mathcal{O}_2) \subseteq \eta$:

1. *Pour tout $r_1 \in \mathcal{R}_{1,0}$, tout $r_2 \in \mathcal{R}_{2,0}$ tels que $(i \Rightarrow_1 r_1)$, $(i \Rightarrow_2 r_2)$ et $(\varphi(r_1) \wedge \varphi(r_2)$ peut être satisfaite), il existe une valuation σ satisfaisant $(\varphi(r_1) \wedge (\varphi(r_2))$. Considérons un modèle avec un état initial q_0 tel que $\upsilon(q_0) = \sigma$. D'après la condition 1*

de la définition 36, nous avons $(q_0, r_1) \in \varrho(\mathcal{M}, \mathcal{O}_1)$ *et* $(q_0, r_2) \in \varrho(\mathcal{M}, \mathcal{O}_2)$. η *étant définie comme l'union de la relation de simulation pour tous les modèles, nous avons* $(r_1, r_2) \in \eta$ *par construction.*

2. *Supposons que* P_n *est vérifiée : pour tout* $r_1 \in \mathcal{R}_{1,n}, r_2 \in \mathcal{R}_{2,n}, (r_1, r_2) \in \varrho(\mathcal{O}_1, \mathcal{O}_2) \Longrightarrow (r_1, r_2) \in \eta$. *Pour tout* $(r_1, r_2) \in \eta(\mathcal{O}_1, \mathcal{O}_2)$, *tout* $r'_1 \in R_1$ *et tout* $r'_2 \in R_2$ *tels que* $r_1 \Rightarrow_1 r'_1$, $r_2 \rightarrow r'_2$ *et* $(\varphi(r'_1) \wedge \varphi(r'_2))$ *peut être satisfait, il existe une valuation* σ *qui satisfait* $(\varphi(r'_1) \wedge (\varphi(r'_2))$. *Par construction,* $(r_1, r_2) \in \eta(\mathcal{O}_1, \mathcal{O}_2)$ *seulement s'il existe un modèle* $\mathcal{M}$, *et* $q \in Q$ *pour ce modèle* $\mathcal{M}$, *tels que* $(q, r_1) \in \varrho(\mathcal{M}, \mathcal{O}_1)$ *et* $(q, r_2) \in \varrho(\mathcal{M}, \mathcal{O}_2)$. *Si ce n'est pas déjà le cas, il est possible d'ajouter à ce modèle un état* q' *successeur de* q *tel que* $\upsilon(q') = \sigma$. η *est défini comme l'union de la relation de simulation pour tous les modèles qui contenaient ce modèle. Selon la condition 2 de la définition 36, nous avons* $(r'_1, r'_2) \in \eta$ *par construction.*

Nous prouvons maintenant que η *est cohérent. Considérons* $(r_1, r_2) \in \eta$. *Par construction,* (r_1, r_2) *existe dans* η *seulement s'il existe un modèle* $\mathcal{M}$, *et* $q \in Q$ *pour ce modèle* $\mathcal{M}$, *tels que* $(q, r_1) \in \varrho(\mathcal{M}, \mathcal{O}_1)$ *et* $(q, r_2) \in \varrho(\mathcal{M}, \mathcal{O}_2)$. *Il y a deux cas : (a)* $\mathcal{M} \models \mathcal{O}_1$ *et (b)* $\mathcal{M} \not\models \mathcal{O}_1$.

(a) Supposons que $\mathcal{M} \models \mathcal{O}_1$. *Par hypothèse, nous avons également* $\mathcal{M} \models \mathcal{O}_2$. *D'après la définition des simulations consistantes (définition 36) et le théorème de décision pour la relation de satisfaction (théorème 2), nous avons* $r_1 \notin \mathcal{I}_1$ *et* $r_2 \notin \mathcal{I}_2$. *L'axiome de la définition 43 est vérifiée.*

(b) Supposons que $\mathcal{M} \not\models \mathcal{O}_1$. *Dans ce cas, nous ne savons pas si* r_1 *est illégal, ou non. La même observation peut être faite pour* r_2. *Il est donc a priori possible d'avoir* $r_1 \notin \mathcal{I}_1$ *et* $r_2 \in \mathcal{I}_2$. *Supposons que ce soit le cas. Cela signifie que, si* $\mathcal{O}_1$ *est déterministe, il existe un mot* u *reconnu par* $\mathcal{O}_1$ *tel que* $r_1 \in obs(i, u)$ *(et* $obs(i, u) \cap C^I = \emptyset$*), et non reconnu par* $\mathcal{O}_2$ *(car* $(r_1, r_2) \in \eta$ *et* $r_2 \in \mathcal{I}_2$*). Considérons tout modèle dont le langage support serait réduit à ce seul mot. Ce modèle est un modèle de* $\mathcal{O}_1$ *(selon la définition 35), mais pas de* $\mathcal{O}_2$, *ce qui contredit notre hypothèse initiale. Il n'est donc pas possible d'avoir* $r_1 \notin \mathcal{I}_1$ *et* $r_2 \in \mathcal{I}_2$. *Nous vérifions ainsi la condition de la définition 43.*

Ainsi, nous concluons à l'existence d'une simulation satisfaisant les conditions du théorème.

Remarque 8 *La déterminisation complète du moniteur $\mathcal{O}_1$ est nécessaire pour vérifier l'état de cohérence ($\mathcal{O}_1 \preceq \mathcal{O}_2 \Rightarrow \eta$ consistant). En effet, si les moniteurs ne sont pas déterministes, il est possible d'avoir le raffinement de $\mathcal{O}_2$ par $\mathcal{O}_1$, sans vérifier cet axiome.*

7.6 Algèbre de composition

Nous utilisons le concept d'ensemble d'acceptation pour compléter la définition sémantique des moniteurs. Il s'agit de relier chaque état d'un moniteur déterministe (voir la définition 40, section 7.4.3) à un ensemble d'ensembles de valuations observées. Ces ensembles sont en quelque sorte tous les modèles observés à partir d'un état donné. En toute rigueur, différencier les états illégaux ou non est nécessaire. Il est raisonnable de s'en dispenser en considérant que les ensembles d'acceptation définissent les comportements observés des modèles, acceptés ou non *in fine*. Cela peut se faire de différentes manières sans être fondamental dans ce que nous présentons ci-dessous.

Considérons la fonction φ avec l'opérateur diamant ($\diamond$). Cette fonction peut être décomposée pour séparer la partie principale, sans diamant, et les différentes formules sous diamant : $\forall r \in R, \varphi(r) = \varphi_0(r) \wedge \diamond\varphi_1(r) \wedge ... \wedge \diamond\varphi_n(r)$. Cette décomposition permet de définir les ensembles d'acceptation pour les états du moniteur.

Définition 44 *Ensemble d'acceptation*
Soit $\mathcal{O} = (\Pi, R, i, \Rightarrow, \mathcal{I}, \varphi)$ un moniteur, et C un ensemble de configurations sur $\mathcal{O}$. Un ensemble d'acceptation est un ensemble d'ensembles de valuation. La fonction $Acc : R \rightarrow \mathcal{P}(\mathcal{P}(2^{\Pi}))$ associe un ensemble d'acceptation à chaque état d'un moniteur. Pour tout $r \in R$, $Acc(r) = \{X\}$ tel que :

1. $\forall x \in X, \forall r' \in R, r \rightarrow r' \Rightarrow x \models \varphi(r')$
2. $\forall \diamond \varphi_{k,k>1}(r), \exists x \in X, x \models \varphi_k(r)$

À partir de la définition 44, les ensembles d'acceptation sont étendus pour définir des ensembles d'acceptation convexes en calculant les limites inférieure et supérieure de ces ensembles. Soit $\Psi(r) = \bigwedge_{r \rightarrow r'} \varphi(r')$, la première condition de la définition 44 est équivalente à $\forall x \in X, x \models \Psi(r)$.

Nous définissons les ensembles φ^- et φ^+ tels que $\varphi^-(r) = \{\{x\}\}_{x \models \varphi(r)}$ et $\varphi^+(r) = \{\{x\}_{x \models \Psi(r)}\}$. Nous considérons également une opération $\oplus$ telle que $\varphi_1 \oplus \varphi_2 = \{X \cup Y | X \subseteq \varphi_1, Y \subseteq \varphi_2\}$. Pour tout $r \in R$, les ensemble d'acceptation convexe des moniteurs sont définis par $Acc(r) = [\varphi_1^-(r) \oplus ... \oplus \varphi_n^-(r), \varphi^+(r)]$.

Il est donc possible d'associer des ensembles d'acceptation convexes à chaque état du moniteur et, par conséquent, de transposer un moniteur sous la forme d'un automate d'acceptation convexe équivalent. Les automates d'acceptation ont une algèbre assez complète avec une complexité algorithmique raisonnable [51]. Nous verrons dans le chapitre 8 une extension de ce formalisme prenant en charge les propriétés d'états. La définition de la conjonction, de la composition et du quotient, transposée dans ce formalisme, permet aux moniteurs d'hériter de ces opérations. De plus, à partir d'un automate d'acceptation convexe, il est envisageable de construire un moniteur équivalent en transposant chaque forme d'ensemble convexe adaptée au moniteur sous la forme d'une formule logique comme $\varphi_0(r) \wedge \diamond\varphi_1(r) \wedge ... \wedge \diamond\varphi_n(r)$.

7.7 Discussion

Une théorie de contrats basée sur les moniteurs comme spécification, et dont les réalisations peuvent être des modèles de Kripke, a été définie et dotée d'une algèbre partielle. En particulier, une relation de satisfaction a pu être définie. Il en est de même pour le raffinement. Cependant, la définition des opérations de composition est plus difficile, sauf peut-être pour la conjonction, et reste un problème ouvert. La transposition d'un moniteur sous la forme d'un automate d'acceptation équivalent est une réponse à ce problème. Dans le prochain chapitre, nous proposons de prolonger notre étude dans ce sens, en partant cette fois-ci des automates d'acceptation pour les étendre sous une forme propositionnelle. L'objectif reste de construire un formalisme de spécification prenant en charge les propriétés d'états, tout en le dotant de manière plus convaincante et efficace de bonnes propriétés algébriques.

Chapitre 8

Automates déterministes à ensembles d'acceptation propositionnels

Les automates déterministes à ensembles d'acceptation propositionnels (DPAA [1]) sont proposés pour saisir les exigences de système exprimant les comportements à temps discret obligatoires et interdits. La principale caractéristique de ce formalisme est qu'il peut exprimer le comportement attendu lorsque le système est dans un état particulier.

Les DPAA combinent des propriétés d'état, exprimées sous forme de formules propositionnelles, et des propriétés temporelles simples en temps discret, exprimées sous forme d'actions obligatoires et interdites chaque fois qu'une propriété d'état donnée est vérifiée. Ce formalisme étend les systèmes de transition modaux à un cadre propositionnel, où les modèles sont des structures de Kripke, plutôt que des systèmes de transition étiquetés. Les opérateurs de composition sur DPAA sont fournis, ce qui en fait une théorie d'interface, avec une relation de raffinement, des opérateurs de composition parallèle, de conjonction et de quotient.

Les DPAA sont proposés comme un formalisme de spécification capable de capturer à la fois le comportement d'entrée/sortie et les propriétés d'état d'un composant. Ce formalisme s'accompagne d'une algèbre de composition, complétée par les opérateurs suivants :

Raffinement relation $\preceq$, telle que $A_1 \preceq A_2$ si, et seulement si, pour tout M, $M \models A_1$ implique $M \models A_2$

Conjonction $A_1 \wedge A_2 = \max\{A \text{ tel que } A \preceq A_1 \text{ et } A \preceq A_2\}$

Produit $A_1 \otimes A_2 = \min\{A \text{ tel que, pour tout } M_1,\ M_2,\ M_1 \models A_1 \text{ et } M_2 \models A_2 \text{ implique } M_1 \times M_2 \models A\}$

Quotient $A_1/A_2 = \max\{A \text{ tel que } A \otimes A_2 \preceq A_1\}$.

1. DPAA : Deterministic Propositional Acceptance Automata

Ce chapitre est organisé comme suit. Les automates déterministes à ensembles d'acceptation propositionnels (DPAA) sont introduits dans la section 8.1, en tant que couche théorique des automates au-dessus des ensembles d'acceptation. Les sections 8.2 et 8.3 établissent les relations de satisfaction et de raffinement. L'algèbre de composition est finalement définie dans la section 8.4.

8.1 Syntaxe

Les DPAA permettent de saisir les propriétés d'état, exprimées sous forme de formules propositionnelles, et les variétés de comportements, exprimées sous forme d'ensembles d'acceptation (AS). A l'image des automates moniteurs, les exécutions d'un modèle de Kripke (KM) peuvent être observées. À partir de son état initial, un DPAA observe les exécutions d'un KM, et l'approuve si, pour chaque état de KM, son ensemble d'états successeurs correspond à l'un des AS de la spécification. Chaque état d'un automate donné peut être associé à plusieurs AS, qui correspondent intuitivement à des ensembles de successeurs possibles d'un état observé d'un KM.

Définition 45 *Deterministic Propositional Acceptance Automaton (DPAA)*
Un automate déterministe à ensembles d'acceptation propositionnels (DPAA) sur l'alphabet Π *est un tuple* $\mathcal{A} = (\Pi, R, r_0, \Rightarrow, \varphi, Acc)$ *tel que :*

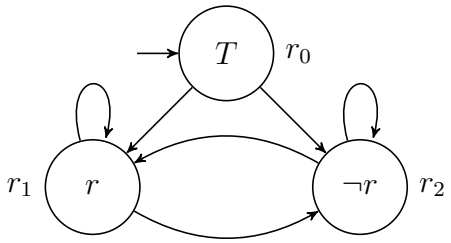
- Π *est un ensemble de proposition atomique ;*
- R *un ensemble d'états :*
- r_0 *l'état initial ;*
- $\Rightarrow \subseteq R \cup \{r_0\} \times R$ *est la relation de transition entre le états ;*
- $\varphi : R \rightarrow \boldsymbol{Prop}(\Pi)$ *est une fonction de valuation associant une formule atomique à chaque état ;*
- $\forall r \in R, \forall r_1, r_2 \in R, r \Rightarrow r_1$ *et* $r \Rightarrow r_2$ *et* $\varphi(r_1) \wedge \varphi(r_2)$ *satisfiables implique que* $r_1 = r_2$ *;*
- $Acc : R \rightarrow 2^{2^{2^{\Pi}}}$ *est une relation associant un ensemble d'acceptation à chaque état.*

Un DPAA est dit réduit si, et seulement si :

1. $\forall r \in R, Acc(r) \neq \emptyset$,
2. $\forall r \in R, \forall \alpha \in Acc(r), \forall \sigma \in \alpha, \exists r' \in R, r \Rightarrow r'$ *et* $\sigma \models \varphi(r')$.

Notons qu'il n'est pas nécessaire que $r_0 \in R$. En particulier, lorsque $r_0 \notin R$, alors l'automate n'a pas besoin d'être déterministe dans son état initial. En outre, l'état initial peut avoir l'ensemble vide comme ensemble d'acceptation dans un DPAA réduit. De même, l'invariance par bégaiement (*stuttering invariant*) n'est pas nécessaire pour les DPAA.

Exemple 36 *La figure 8.1 définit, sous la forme d'un DPAA sur l'alphabet* $\Pi = \{m, r, o\}$*, une règle sur le contrôle des portes d'une navette en fonction de la demande d'arrêt. Les propositions m, r et o indiquent respectivement le mouvement du bus, la requête d'arrêt et l'ouverture de la porte.*



	$Acc(r_i)$
r_0	$2^{2^{2^{\Pi}}}$
r_1	$\{\alpha \mid \forall \sigma \in \alpha, o \in \sigma\}$
r_2	$2^{2^{2^{\Pi}}}$

FIGURE 8.1 – DPAA d'une exigence spécifiant l'ouverture des portes de la navette à la demande d'un passager.

8.2 Relation de satisfaction

Les DPAA permettent d'exprimer des exigences à respecter par des modèles de réalisation (KM). La relation de satisfaction entre KM et DPAA permet de savoir si un KM respecte ces exigences, ou au contraire les viole. Pour décider de l'existence d'une relation de satisfaction entre un KM et un DPAA, nous considérons une relation de simulation entre ces deux formalismes.

Définition 46 *Simulation entre un KM et un DPAA*
La relation de simulation entre un KM $\mathcal{M}$ *et un DPAA* $\mathcal{A}$*, dénommée* $\varrho(\mathcal{M}, \mathcal{A})$*, est la plus faible relation* $\varrho \subseteq Q \times R$ *telle que :*

1. $\forall q_0 \in Q_0, (q_0, r_0) \in \varrho$;
2. $\forall (q, r) \in \varrho$, $\forall q' \in Q, \forall r' \in R$, $q \rightarrow q'$ *et* $r \Rightarrow r'$ *et* $\upsilon(q') \models \varphi(r')$ *implique que* $(q', r') \in \varrho$.

Notons que les conditions de la définition 46 donnent une construction inductive de ϱ. Il existe donc une simulation unique ϱ pour un modèle $\mathcal{M}$ et un DPAA $\mathcal{A}$.

On note $\Pi_{\mathcal{M}}$ l'alphabet pour un KM $\mathcal{M}$, $\Pi_{\mathcal{A}}$ l'alphabet pour un DPAA $\mathcal{A}$. Nous introduisons l'opérateur $\downarrow_{\Pi'}$, qui désigne la restriction d'une valuation σ à la partie relative de l'alphabet Π', pour une valuation basée sur un alphabet Π tel que $\Pi' \subseteq \Pi$.

Définition 47 *Simulation consistante* $\varrho(\mathcal{M}, \mathcal{A})$
Soit $\Pi_{\mathcal{M}}$ *et* $\Pi_{\mathcal{A}}$ *tel que* $\Pi_{\mathcal{A}} \subseteq \Pi_{\mathcal{M}}$. *La simulation* $\varrho(\mathcal{M}, \mathcal{A})$ *est consistante si, et seulement si, pour tout* $(q, r) \in \varrho(\mathcal{M}, \mathcal{A})$, *il existe* $\alpha \in Acc(r)$ *tel que :*

1. $\forall q' \in Q, q \rightarrow q'$ *implique que* $(\upsilon(q') \downarrow_{\Pi_{\mathcal{A}}}) \in \alpha$,
2. $\forall \sigma \in \alpha, \exists q' \in Q$ *tel que* $q \rightarrow q'$ *et* $\sigma = (\upsilon(q') \downarrow_{\Pi_{\mathcal{A}}})$.

La définition 48 donne une procédure pour décider si un KM satisfait ou non à un DPAA sans avoir à calculer les langages support associés. Cette procédure calcule la relation de simulation, avec une complexité quadratique [99].

Définition 48 *Relation de satisfaction entre KM et DPAA*
Un KM $\mathcal{M}$ *est le modèle d'un DPAA* $\mathcal{A}$ *si, et seulement si, la relation de simulation* $\varrho(\mathcal{M}, \mathcal{A})$ *est consistante.*

Un DPAA réduit est dit consistant si, et seulement si, $r_0 \in R$.

8.3 Relation de raffinement

Le raffinement permet d'assurer la cohérence entre plusieurs spécifications pour des niveaux d'abstraction différents.

Définition 49 *Relation de raffinement entre DPAA*
$\mathcal{A}_1$ *est un raffinement de* $\mathcal{A}_2$, *noté* $\mathcal{A}_1 \preceq \mathcal{A}_2$, *si, et seulement si, pour tout modèle* $\mathcal{M}$, $\mathcal{M} \models \mathcal{A}_1$ *implique* $\mathcal{M} \models \mathcal{A}_2$.

Il est possible de définir une relation de simulation entre deux DPAA. Cela est utile pour déterminer si un automate donné $\mathcal{A}_1$ est un raffinement d'un autre automate $\mathcal{A}_2$ sans avoir à considérer tous les modèles possibles. Le point important de cette relation implique de considérer le domaine d'observation de chaque paire d'états, r_1 de $\mathcal{A}_1$ et r_2 de $\mathcal{A}_2$. Un état r_1 de $\mathcal{A}_1$ est lié à un état r_2 de $\mathcal{A}_2$ si le domaine d'observation de r_1 intercepte le domaine d'observation de r_2, en ce sens qu'une série de modèles (ou de mots) observée par r_1 serait également observée par r_2.

Définition 50 *Relation de simulation entre DPAA*
La relation de simulation entre deux DPAA $\mathcal{A}_1$ et $\mathcal{A}_2$, notée $\eta(\mathcal{A}_1, \mathcal{A}_2)$, est la plus petite relation $\eta \subseteq \mathcal{R}_1 \times \mathcal{R}_2$ tel que :

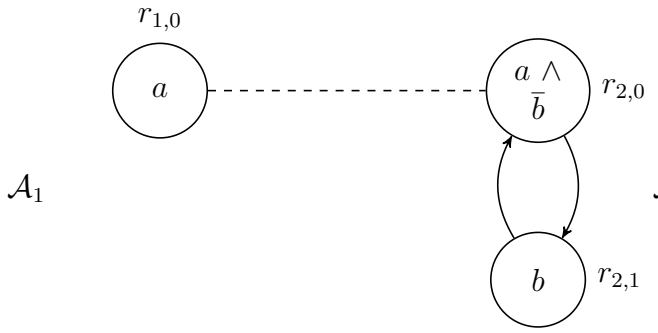
1. $(r_{1,0}, r_{2,0}) \in \eta$
2. $\forall (r_1, r_2) \in \eta$, $\forall r'_1 \in R_1, \forall r'_2 \in \mathcal{R}_2$, $(r_1 \Rightarrow_1 r'_1)$ *et* $(r_2 \Rightarrow_2 r'_2)$
   et $(\exists \alpha \in Acc_1(r_1), \exists \sigma \in \alpha, \sigma \models \varphi_1(r'_1) \wedge \varphi_2(r'_2))$ *implique* $(r'_1, r'_2) \in \eta$

La définition 51 formalise la notion de simulation consistante, utile pour démonter le raffinement.

Définition 51 *Simulation consistante entre DPAA*
La simulation $\eta(\mathcal{A}_1, \mathcal{A}_2)$ est dite consistante si, et seulement si, la propriété suivante est vérifiée :

— $\forall (r_1, r_2) \in \eta, Acc(r_1) \subseteq Acc(r_2)$.

La figure 8.2 montre une représentation d'une relation de simulation (en pointillés) entre deux DPAA $\mathcal{A}_1$ (à gauche) et $\mathcal{A}_2$ (à droite), et un tableau donnant les AS associés aux états pour chacun de ces automates.



	$Acc(\mathcal{A}_1)$	$Acc(\mathcal{A}_2)$
$r_{1,0}$	$\{a\overline{b}\}$	
$r_{2,0}$		$\{a\overline{b}, ab\}, \{a\overline{b}\}$
$r_{2,1}$		$\{\overline{a}b\}$

FIGURE 8.2 – Simulation entre $\mathcal{A}_1$ (à gauche) et $\mathcal{A}_2$ (à droite)

La relation de simulation est directement calculable. Sur cette base, et en s'appuyant sur la définition 50, il est finalement possible de proposer un théorème pour décider du raffinement entre deux DPAA.

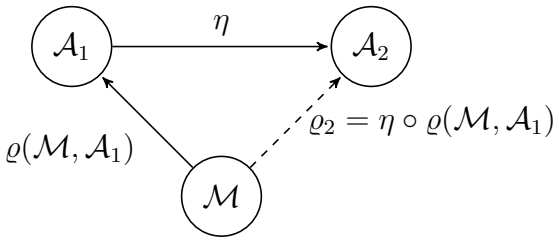
Théorème 4 *Théorème du raffinement*
$\mathcal{A}_1 \preceq \mathcal{A}_2$ si, et seulement si, la simulation $\eta(\mathcal{A}_1, \mathcal{A}_2)$ est consistante (définition 50).

Preuve 12

Soit $\mathcal{A}_1$ et $\mathcal{A}_2$ deux DPAA. Soit $\eta(\mathcal{A}_1, \mathcal{A}_2)$ la simulation de $\mathcal{A}_1$ vers $\mathcal{A}_2$ satisfaisant les conditions de la définition 50. Nous considérons les simulations $\varrho(\mathcal{M}, \mathcal{A}_1)$ et $\varrho(\mathcal{M}, \mathcal{A}_2)$ entre un modèle $\mathcal{M}$ et respectivement $\mathcal{A}_1$, $\mathcal{A}_2$.

Implication 1 : $\eta(\mathcal{A}_1, \mathcal{A}_2)$ consistante implique $\mathcal{A}_1 \preceq \mathcal{A}_2$

Soit $\mathcal{M}$ un modèle tel que $\mathcal{M} \models \mathcal{A}_1$. D'après la définition 47, on sait que la relation de simulation $\varrho(\mathcal{M}, \mathcal{A}_1)$ est consistante. On considère la relation $\varrho_2 = \eta(\mathcal{A}_1, \mathcal{A}_2) \circ \varrho(\mathcal{M}, \mathcal{A}_1)$. Nous avons $\varrho_2 \subseteq Q \times R_2$ par construction. Nous allons prouver que $\varrho(\mathcal{M}, \mathcal{A}_2) \subseteq \varrho_2$ et que ϱ_2 est consistante. Cela prouvera que $\varrho(\mathcal{M}, \mathcal{A}_2)$ est consistante, et donc que $\mathcal{M} \models \mathcal{A}_2$.



On considère des ensembles supports $\mathcal{R}_n$ de $\mathcal{A}_2$ tels que :

1. $\mathcal{R}_0 = \{r_{2,0}\}$,
2. $\mathcal{R}_{n+1} = \{r' \mid r \Rightarrow_2 r' \wedge r \in \mathcal{R}_n\} \cup \mathcal{R}_n$.

Remarquons que nous avons, par construction, $\forall n, \mathcal{R}_n \subseteq R_2$ et il existe m tel que $\mathcal{R}_m$ contient tous les états atteignables de $\mathcal{A}_2$ (et donc, $\forall n > m, \mathcal{R}_n = \mathcal{R}_m$).

Considérons la propriété suivante P_n : $\forall r_2 \in \mathcal{R}_n, (q, r_2) \in \varrho(\mathcal{M}, \mathcal{A}_2)$ implique que $(q, r_2) \in \varrho_2$. Nous prouvons que P_n est vrai pour tout n, et donc que $\varrho(\mathcal{M}, \mathcal{A}_2) \subseteq \varrho_2$:

1. *D'après la définition 46, condition 1, $\forall q_0 \in Q_0$, nous avons $(q_0, r_{1,0}) \in \varrho(\mathcal{M}, \mathcal{A}_1)$ et $(q_0, r_{2,0}) \in \varrho(\mathcal{M}, \mathcal{A}_2)$. D'après la condition 1 de la définition 50, nous avons $(r_1, r_2) \in \eta(\mathcal{A}_1, \mathcal{A}_2)$ par construction, et $(q_0, r_{2,0}) \in \varrho_2$ car $(r_1, r_2) \circ (q_0, r_1) = (q_0, r_2)$, ce qui prouve que P_0 est vérifiée.*
2. *Supposons que P_n est vérifiée : $\forall r_2 \in \mathcal{R}_n$, nous avons $(q, r_2) \in \varrho(\mathcal{M}, \mathcal{A}_2)$ implique que $(q, r_2) \in \varrho_2$. D'après la condition 2 de la définition 46, $\forall (q, r_2) \in \varrho(\mathcal{M}, \mathcal{A}_2)$, $\forall q' \in Q$, $\forall r_2' \in R_2$ tel que $q \to q'$ et $r_2 \Rightarrow_2 r_2'$, nous avons $(q', r_2') \in \varrho(\mathcal{M}, \mathcal{A}_2)$ si $\upsilon(q') \models \varphi(r_2')$. De l'hypothèse que $\mathcal{M}$ est un modèle de $\mathcal{A}_1$, nous déduisons que la relation de simulation $\varrho(\mathcal{M}, \mathcal{A}_1)$ est consistante et qu'il existe un ensemble $\alpha \in Acc(r_1)$, et*

une valuation $\sigma \in \alpha$ tels que $\upsilon(q') \models \sigma$ (pour $q' \in Q, q \to q'$). D'après l'hypothèse que $\mathcal{A}_1$ est consistante, il existe nécessairement un état $r'_1 \in R_1$ tel que $r_1 \Rightarrow_1 r'_1$ et $\exists \alpha \in Acc(r_1), \exists \sigma \in \alpha, \upsilon(q') \models \sigma$, donc $\upsilon(q') \models \varphi(r'_1)$. D'après la condition 2 de la définition 50, nous avons $(r'_1, r'_2) \in \eta(\mathcal{A}_1, \mathcal{A}_2)$ par construction, et $(q', r'_2) \in \varrho_2$ car $(r'_1, r'_2) \circ (q', r'_1) = (q', r'_2)$. Nous vérifions donc que $\forall r'_2 \in \mathcal{R}_{n+1}, (q', r'_2) \in \varrho(\mathcal{M}, \mathcal{A}_2)$ implique que $(q, r_2) \in \varrho_2$, prouvant ainsi que la propriété P_{n+1} est vérifiée.

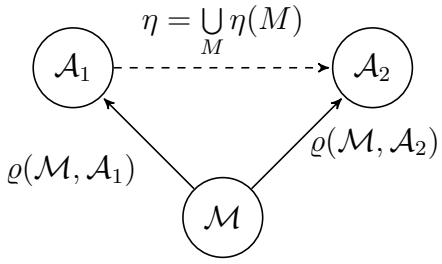
Nous en concluons donc que P_n est vérifiée pour tout n et que $\varrho(\mathcal{M}, \mathcal{A}_2) \subseteq \varrho_2$.
Nous allons maintenant prouver que ϱ_2 est consistant :

— *Nous avons $\forall (r_1, r_2) \in \eta, Acc(r_1) \subseteq Acc(r_2)$ (définition 51). $\forall (q, r_1) \in \varrho$, il existe $\alpha \in Acc(r_1)$ vérifiant les conditions de la définition 46. Par construction, pour tout $(q, r_2) \in \varrho_2$, il existe r_1 tel que $(r_1, r_2) \in \eta(\mathcal{A}_1, \mathcal{A}_2)$ et $(q, r_1) \in \varrho(\mathcal{M}, \mathcal{A}_1)$. D'après la définition 51, cela garantit qu'il existe un ensemble $\alpha \in Acc(r_2)$ vérifiant les conditions de la définition 46 pour tout $(q, r_2) \in \varrho$.*

Nous en concluons que $\varrho(\mathcal{M}, \mathcal{A}_2)$ est également consistant, et donc que $\mathcal{M} \models \mathcal{A}_2$.

Implication 2 : $\mathcal{A}_1 \preceq \mathcal{A}_2$ implique que $\eta(\mathcal{A}_1, \mathcal{A}_2)$ est consistante

Soit $\mathcal{M}$ un modèle. On considère une relation $\eta(\mathcal{M}) = \varrho(\mathcal{M}, \mathcal{A}_2) \circ \varrho(\mathcal{M}, \mathcal{A}_1)^{-1}$. Nous savons que $\varrho(\mathcal{M}, \mathcal{A}_2) \subseteq Q \times R_2$, $\varrho(\mathcal{M}, \mathcal{A}_1) \subseteq Q \times R_1$. Par conséquent, nous avons $\varrho(\mathcal{M}, \mathcal{A}_1)^{-1} \subseteq R_1 \times Q$ et $\eta(\mathcal{M}) \subseteq R_1 \times R_2$. Il est possible de définir η comme l'union de toutes les relations $\eta(M)$: $\eta = \bigcup_{\mathcal{M}} \eta(\mathcal{M})$. Nous allons prouver que $\eta(\mathcal{A}_1, \mathcal{A}_2) \subseteq \eta$ et que η est consistante. Cela prouvera que $\eta(\mathcal{A}_1, \mathcal{A}_2)$ est aussi consistant.



Considérons les ensembles support $\mathcal{R}_{1,n}$ de $\mathcal{A}_1$ et $\mathcal{R}_{2,n}$ de $\mathcal{A}_2$ tels que, pour $i = 1, 2$:

1. $\mathcal{R}_{i,0} = \{r_{i,0}\}$,
2. $\mathcal{R}_{i,n+1} = \{r' \mid r \Rightarrow_i r' \wedge r \in \mathcal{R}_{i,n}\} \cup \mathcal{R}_{i,n}$.

On peut remarquer que, par construction, $\forall n, \mathcal{R}_{i,n} \subseteq R_i$ *et il existe* m *tel que* $\mathcal{R}_{i,m}$ *contient tous les états accessibles de* $\mathcal{A}_i$ *(et donc,* $\forall n > m, \mathcal{R}_{i,n} = \mathcal{R}_{i,m}$*). De plus, il est possible de faire correspondre n'importe quel ensemble* R_n *avec un modèle de référence* $\mathcal{M}$ *tel que* $\mathcal{M} \models \mathcal{A}_1$*. Ce modèle est construit par extension à partir de l'état initial, et en choisissant à chaque fois un ensemble d'acceptation* α*, chaque* $\sigma \in \alpha$ *étant associé à un nouvel état du modèle.*

Considérons alors la propriété suivante P_n *:* $\forall r_1 \in \mathcal{R}_{1,n}, r_2 \in \mathcal{R}_{2,n}, (r_1, r_2) \in \varrho(\mathcal{A}_1, \mathcal{A}_2)$ *implique que* $(r_1, r_2) \in \eta$*. Nous prouvons que* P_n *est vrai pour tout* n*, et donc que* $\eta(\mathcal{A}_1, \mathcal{A}_2) \subseteq \eta$ *:*

1. *Considérons des modèles quelconques. Soit* Q_0 *l'ensemble des états initiaux pour ces modèles. D'après la condition 1 de la définition 46, nous avons* $(q_0, r_{1,0}) \in \varrho(\mathcal{M}, \mathcal{A}_1)$ *et* $(q_0, r_{2,0}) \in \varrho(\mathcal{M}, \mathcal{A}_2)$ *pour tout* $q_0 \in Q_0$*. La relation* η *étant définie comme l'union des relations de simulation pour tous les modèles, nous avons* $(r_{1,0}, r_{2,0}) \in \eta$ *par construction.*

2. *Supposons que* P_n *est vérifiée :* $\forall r_1 \in \mathcal{R}_{1,n}, r_2 \in \mathcal{R}_{2,n}, (r_1, r_2) \in \varrho(\mathcal{A}_1, \mathcal{A}_2)$ *implique* $(r_1, r_2) \in \eta$*. Considérons* $(r_1, r_2) \in \eta(\mathcal{A}_1, \mathcal{A}_2) \cap (\mathcal{R}_{1,n} \times \mathcal{R}_{2,n})$*. D'après la condition 2 de la définition 50,* $\forall r'_1 \in R_1$*,* $\forall r'_2 \in R_2$ *tels que* $r_1 \Rightarrow_1 r'_1$*,* $r_2 \rightarrow r'_2$*,* $(r'_1, r'_2) \in \eta(\mathcal{A}_1, \mathcal{A}_2)$ *implique qu'il existe un ensemble* $\alpha \in Acc_1(r_1)$ *et une valuation* $\sigma \in \alpha$ *qui satisfait* $(\varphi(r'_1) \wedge (\varphi(r'_2))$*. Partant du modèle de référence* $\mathcal{M}$ *de* R_n*, il est possible de construire par extension un modèle* $\mathcal{M}'$ *tel qu'il existe un état* $q \in Q$*, avec* $readyset_{\mathcal{M}'}(q) = \alpha$*. Dans ce cas, nous avons* $(r'_1, r'_2) \in \eta(\mathcal{M}')$*. La relation* η *est définie comme l'union des relations de simulation pour tous les modèles, ce qui inclut ce modèle. D'après la condition 2 de la définition 46, nous avons* $(r'_1, r'_2) \in \eta$ *par construction.*

Nous allons maintenant prouver que η *est consistant. Considérons* $(r_1, r_2) \in \eta$*. Supposons par l'absurde que* $Acc(r_1) \nsubseteq Acc(r_2)$*. Cela signifie qu'il existerait* σ *tel que* $\sigma \in Acc(r_1)$ *et* $\sigma \notin Acc(r_2)$*. A partir de* r_0*, construisons par extension un modèle* $\mathcal{M}$ *en choisissant pour chaque état* q *de ce modèle un ensemble d'acceptation tel que* $Ready_{\mathcal{M}}(q)$*, avec un état* q_1 *tel que* $Ready_{\mathcal{M}(q_1)} = \sigma$*. Notre hypothèse initiale implique que* $M \models A_2$*, et donc* $\sigma \in \alpha(r_2)$ *(condition 2.a de la définition 46). On en déduit que* $\sigma \in \alpha(r_1)$ *et* $\sigma \in \alpha(r_2)$*, ce qui implique que* $\alpha(r_1) \subseteq \alpha(r2)$*.*

Ainsi, on conclut à l'existence d'une simulation remplissant les conditions du théorème.

8.4 Algèbre de composition

Les opérateurs de composition sont une adaptation des opérateurs définis sur des modèles tels que les interfaces modales. L'alphabet, la structure des automates et les formules d'état sont obtenus de manière similaire à la conjonction, au produit et au quotient : l'alphabet par union, la structure par produit cartésien et les formules par conjonction. Les états des modèles observés sont les mêmes pour ces trois opérations. La différence réside dans les ensembles d'acceptation. Ceux-ci sont obtenus en appliquant les opérations correspondantes de l'algèbre des ensembles d'acceptation (chapitre 6), avec une petite adaptation supplémentaire pour le quotient. La cohérence de ces théorèmes avec les axiomes d'une théorie des interfaces est garantie par construction. Quelques détails sont donnés ci-dessous.

8.4.1 Conjonction

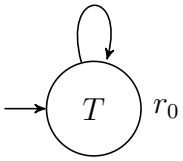
Les exigences consistent à spécifier les propriétés attendues du système en cours de conception. Elles constituent un moyen pour les fabricants d'origine d'interagir avec les fournisseurs. et l'interprétation d'un document d'exigences est la conjonction de toutes ces implications. Ce même concept devrait également être valable pour définir la combinaison de différents points de vue d'exigences telles que la correction fonctionnelle, la sécurité ou l'énergie, sur la base de différents cadres de modélisation qui interagissent.

Théorème 5 *La conjonction $\mathcal{A}_1 \wedge \mathcal{A}_2$ est le DPAA $\mathcal{A} = (\Pi, R, r_0, \Rightarrow, \varphi, Acc)$ tel que :*

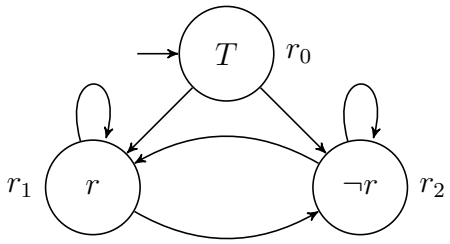
- $\Pi = \Pi_2 \cup \Pi_1$;
- $R = R_1 \times R_2$;
- $r_0 = (r_{1,0}, r_{2,0})$;
- $(r_1, r_2) \Rightarrow (r'_1, r'_2)$ *si, et seulement si,* $r_1 \Rightarrow_1 r'_1$ *et* $r_2 \Rightarrow_2 r'_2$;
- $\varphi(r_1, r_2) = \varphi_1(r_1) \ \wedge \ \varphi_2(r_2)$;
- $Acc(r_1, r_2) = Acc_1(r_1) \wedge Acc_2(r_2)$.

Example. Les DPAA $\mathcal{A}_1$ et $\mathcal{A}_2$ se rapportent à l'exemple de la navette (figure 8.3). Ils traduisent deux exigences, l'interdiction d'ouverture de la porte en cas de mouvement du bus ($\mathcal{A}_1$) et l'ouverture souhaitée à la demande du passager ($\mathcal{A}_2$).

La figure 8.4 montre la conjonction de $\mathcal{A}_1$ et $\mathcal{A}_2$.

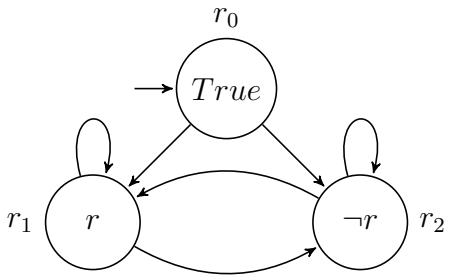


	$Acc(r_i)$
r_0	$\{\alpha \mid \forall \sigma \in \alpha, m \in \sigma \Rightarrow o \notin \sigma\}$



	$Acc(r_i)$
r_0	$2^{2^{2^{\Pi}}}$
r_1	$\{\alpha \mid \forall \sigma \in \alpha, o \in \sigma\}$
r_2	$2^{2^{2^{\Pi}}}$

FIGURE 8.3 – $\mathcal{A}_1$ traduit l'interdiction d'ouverture de la porte si la navette est en mouvement, tandis que $\mathcal{A}_2$ représente l'obligation d'ouverture de la porte à la demande du passager.



	$Acc(r_i)$
r_0	$\{\alpha \mid \alpha \models \Box(m \Rightarrow \neg o)\}$
r_1	$\{\alpha \mid \alpha \models \Box o \wedge (m \Rightarrow \neg o)\}$ $= \{\alpha \mid \alpha \models \Box(o \wedge \neg m)\}$
r_2	$\{\alpha \mid \alpha \models \Box(m \Rightarrow \neg o)\}$

FIGURE 8.4 – $\mathcal{A}_3 = \mathcal{A}_1 \wedge \mathcal{A}_2$

8.4.2 Produit

Chaque fournisseur peut être amené à travailler sur un cahier des charges spécifique. Pour chacun de ces sous-systèmes, un contrat peut être défini. L'intégration des différents composants aboutit à la composition des sous-systèmes définissant l'architecture principale, ce qui se traduit par la composition parallèle des contrats.

Théorème 6 *Le produit* $\mathcal{A}_1 \otimes \mathcal{A}_2$ *est le DPAA* $\mathcal{A} = (\Pi, R, r_0, \Rightarrow, \varphi, Acc)$ *tel que :*

- $R = R_1 \times R_2$;
- $r_0 = (r_{1,0}, r_{2,0})$;
- $(r_1, r_2) \Rightarrow (r'_1, r'_2)$ *si, et seulement si,* $r_1 \Rightarrow_1 r'_1$ *et* $r_2 \Rightarrow_2 r'_2$;
- $\varphi(r_1, r_2) = \varphi_1(r_1) \ \wedge \ \varphi_2(r_2)$;
- $Acc(r_1, r_2) = Acc_1(r_1) \otimes Acc_2(r_2)$.

Exemple 37 *Nous illustrons cet opérateur en considérant les DPAA* $\mathcal{A}_1$ *et* $\mathcal{A}_2$ *tels que* $\mathcal{A}_1$ *exprime que b n'est possible qu'après a, tandis que* $\mathcal{A}_2$ *exprime que b est nécessairement suivi de a (Fig. 8.5).*

La figure 8.6 correspond à l'application du produit sur les DPAA $\mathcal{A}_4$ *et* $\mathcal{A}_5$.

8.4.3 Quotient

L'opération de quotient est l'adjoint à droite de l'opérateur du produit. Soit $\mathcal{A}_1$ et $\mathcal{A}_2$ deux automates d'acceptation (DPAA). Le quotient $\mathcal{A}_1/\mathcal{A}_2$ caractérise les réalisations $\mathcal{M}$ tel que toute réalisation de $\mathcal{A}_2$ composée avec $\mathcal{M}$ est une réalisation de $\mathcal{A}_1$. Elle est déterminée par l'équation suivante :

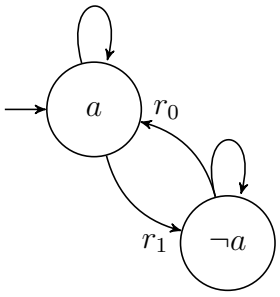
$$\mathcal{A}_1/\mathcal{A}_2 = max\{A \mid A2 \times A \leq A\}.$$

L'opération de quotient peut être exploitée dans une approche de raisonnement compositionnel, pour décomposer une exigence à l'échelle du système en spécifications au niveau des composants, ou chaque fois que la réutilisation des composants est recherchée.

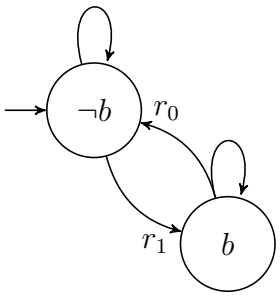
Le quotient de deux DPAA est calculé comme suit :

Théorème 7 *Le quotient* $\mathcal{A}_1/\mathcal{A}_2$ *est le DPAA* $\mathcal{A} = (\Pi, R, r_0, \Rightarrow, \varphi, Acc)$ *tel que :*

- $R = (R_1 \times R_2) \cup R^\tau$;
- $R^\tau = \{\tau_{r_1,r_2} \mid r_1 \in R_1, r_2 \in R_2\} \cup \{\tau\}$;



	$Acc(r_i)$
r_0	$\{a\}, \{a, ab\}, \{a, ab, b\}, \{ab\}, \{ab, b\}, \{b\}, \emptyset$
r_1	$\{a\}, \emptyset$



	$Acc(r_i)$
r_0	$2^{2^{2^\Pi}}$
r_1	$\{a\}, \{a, ab\}, \{ab\}$

FIGURE 8.5 – $\mathcal{A}_4$ exprime que b est possible seulement après a, tandis que $\mathcal{A}_5$ exprimes que b est nécessairement suivi par a.

— $r_0 = (r_{1,0}, r_{2,0})$;

— $(r_1, r_2) \Rightarrow (r'_1, r'_2)$ *si, et seulement si,* $r_1 \Rightarrow_1 r'_1$ *et* $r_2 \Rightarrow_2 r'_2$;

— $\forall r_1 \in R_1, r_2 \in R_2, (r_1, r_2) \Rightarrow \tau_{r_1,r_2}$ *et* $\tau_{r_1,r_2} \Rightarrow \tau$;

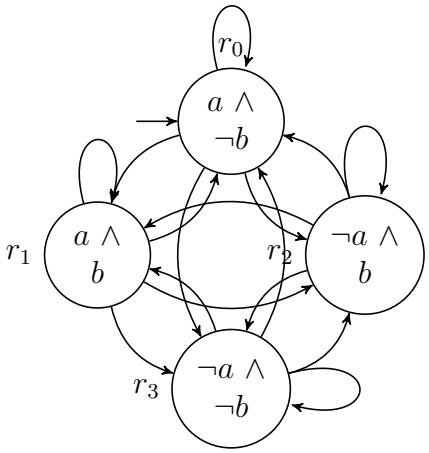
— $\varphi(r_1, r_2) = \varphi_1(r_1) \ \wedge \ \varphi_2(r_2)$;

— $\varphi(\tau_{r_1,r_2}) = \bigvee_{r_1 \Rightarrow_1 r'_1} \varphi_1(r'_1) \ \wedge \bigvee_{\forall \alpha \in Acc(r_1,r_2), \sigma \notin \alpha} \sigma$;

— $\varphi(\tau) = True$;

— $Acc(r_1, r_2) = Acc_1(r_1)/Acc_2(r_2)$;

— $Acc(\tau_{r_1,r_2}) = Acc(\tau) = 2^{2^{2^\Pi}}$.



	$Acc(r_i)$
r_0	$\{a\}, \{a, ab\}, \{a, ab, b\}, \{ab\}, \{ab, b\}, \{b\}, \emptyset$
r_1	$\{a\}, \{a, ab\}, \{ab\}$
r_2	$\{a\}$
r_3	$\{a\}, \emptyset$

FIGURE 8.6 – $\mathcal{A}_6 = \mathcal{A}_4 \otimes \mathcal{A}_5$

Chapitre 9

Systèmes étudiés de bout en bout

Nous proposons d'illustrer les concepts exposés sur deux exemples mettant en œuvre une chaîne de traitement complète. Les données du premier exemple proviennent d'un cas d'étude spécifiant un système de contrôle d'accès pour un parking de voitures [4]. Le second cas d'étude a été constitué pour illustrer l'application des principes présentés pour des systèmes régis par une procédure. Nous nous appuyons sur plusieurs expérimentations [100, 101], et une première intégration formelle avec l'outil Mica [16]. Mica est une bibliothèque OCaml qui permet de définir des systèmes complexes en exploitant l'algèbre des interfaces modales [13]. Dans cette application, la formalisation des exigences a été réalisée manuellement. Notre intention est d'aller plus loin, en déroulant toute la chaîne de traitement à partir des exigences exprimées en langue naturelle.

Concrètement, l'architecture envisagée forme une chaîne de traitement intégrant différentes briques logicielles. Dans un premier temps, le processus vise à construire une représentation abstraite des énoncés en capturant leur sémantique. Une structure pivot est utilisée dans ce but. Les graphes AMR [46] sont un choix de représentation approprié. Le second temps consiste à s'appuyer sur cette représentation pour générer des déclarations formelles dans un format adapté. Cette étape se traduit par la mise en œuvre des principes d'analyse par transduction sémantique. Finalement, la composition des exigences formelles permet d'aboutir à une modélisation des spécifications qui s'inscrit dans un cadre de raisonnement apportant les garanties attendues.

La mise en œuvre de cette chaîne de traitements intègre plusieurs prototypes, dont l'implémentation est effectuée avec les langages OCaml, Python et Java. Le module principal est écrit en Python. Il permet de contrôler les différentes opérations. Les étapes de pré-traitement, qui comprennent le découpage des éléments de chaque phrase en unités syntaxiques (token) et l'étiquetage grammatical (POS tagging), sont réalisés avec l'outil Stanford CoreNLP [48]. Les structures AMR résultent ensuite de l'analyse syntaxico-sémantique des énoncés segmentés et étiquetés. Nous proposons deux algorithmes, dont la mise en œuvre est une adaptation des outils sous licence libre CAMR [82] et STOG [49].

Notre principale contribution porte sur le module ATP (Abstract Transduction Process), developpé en Java. Il intègre les étapes de transduction sémantique, détaillées dans le chapitre 4. Le paramétrage de l'outil comprend la définition d'une ontologie (module OntoMap), utilisée pour simplifier le traitement, et la définition de schémas de transduction (module TransductionPattern), utilisée lors de l'étape d'analyse sémantique abstraite. En l'état actuel, seul un fragment du corpus PropBank [50], dont dépend le corpus AMR, est pris en compte. Ce fragment est caractérisé par les phénomènes linguistiques couverts. Nous avons mis l'accent, dans nos expérimentations, sur les phénomènes présents dans l'énoncé d'une exigence, tels que les modalités déontiques et temporelles, les connecteurs logiques et la négation. Ce fragment est à priori suffisant pour obtenir une bonne couverture des documents de spécification. Il est tout à fait possible de l'étendre pour améliorer cette dernière.

Finalement, nous montrons comment vérifier la consistance d'une spécification en exploitant l'algèbre de composition d'une théorie d'interface. Les dernières étapes du processus ont été déroulées à la main sur quelques exemples pour montrer la faisabilité à priori d'une implémentation. En complément, la pertinence des concepts avancés peut être observée en utilisant l'outil Mica, traduction de l'algèbre des interfaces modales [13].

Nos expérimentations ont été menées sur différents cas d'étude, dont le système de contrôle d'accès d'un garage de voitures (section 9.1) et la manipulation de poupées russes (section 9.2), que nous reprenons dans la suite de ce chapitre. Le premier cas montre l'analyse d'une spécification complète, en passant par deux représentations pivot (graphe AMR et format MPC). Le deuxième cas complète notre étude, avec une seule représentation pivot (graphe AMR) et des calculs par transduction avec plusieurs attributs. La démarche proposée est ici adaptée pour modéliser formellement une procédure définie par un ensemble de règles.

9.1 Système de contrôle d'accès d'un garage de voitures

Spécification informelle du système

Le cas d'étude définit un système relativement classique de contrôle d'accès pour un garage de voitures [4]. Le système est composé de plusieurs entités. Les barrières de sécurité permettent de contrôler les entrées et sorties du parking. La capacité maximale

est fixée. Un ticket est fourni en entrée. Ce ticket autorise la sortie après paiement.

La figure 9.1 donne une vue d'ensemble de l'architecture du système, dont la spécification générale est constituée de trois sections. La section *Gate* intègre les exigences qui caractérisent le comportement des barrières d'accès au parking. Ces exigences peuvent être instanciées pour chaque barrière (en entrée ou sortie). La section *Payment* précise le comportement de la machine de paiement. Finalement, la section *Supervisor* rassemble les exigences se rapportant à l'ensemble du système.

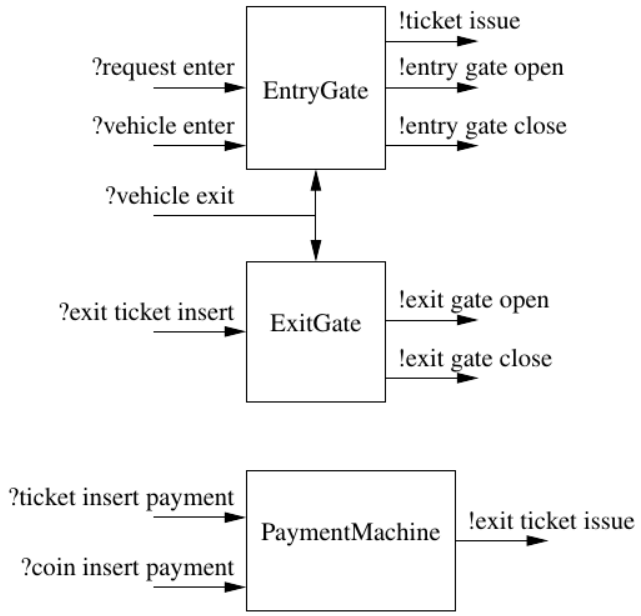


FIGURE 9.1 – Architecture du système "The Parking Garage" [4]

Ce système a été étudié en détail, du point de vue ingénierie des exigences, dans le livre *Contract for System Design* [4]. Nous reprenons quelques exigences, rassemblées dans le tableau 9.1. Elles sont exprimées avec un langage naturel, à savoir l'anglais. Il n'y a aucune contrainte particulière sur la forme de l'énoncé. L'objectif fixé est l'interprétation automatique de ces énoncés, c'est à dire la construction d'une certaine structure formelle. Le but est de démontrer formellement la consistance, ou l'inconsistance, de cet ensemble. En cas d'inconsistance, il s'agit de mettre en évidence les exigences contradictoires.

Ces exigences contiennent plusieurs caractéristiques linguistiques qui doivent être prises en compte. Elles peuvent décrire des enchaînements d'événements, avec des modalités temporelles (*after*, *before*). Elles peuvent également contenir des éléments qui font

Id	*Exigences*
E_1	When a ticket had been issued, the entry gate must open.
E_2	The passage of a car is prohibited if the security gate is closed.
E_3	After an exit ticket is inserted, the exit gate must open.
E_4	If the parking is full or a vehicle is passing, the entry gate should not open.

TABLE 9.1 – Quelques exigences d'un système d'accès pour un parking de voitures

référence à des entités de l'environnement du système (*vehicle*, *entry gate*), ou des informations implicites. Les spécifications contiennent aussi des modalités d'obligation et d'interdiction (*may*, *must*, *cannot*). L'extraction des informations nécessaires pour le passage vers une spécification formelle implique une bonne évaluation de ces différents aspects.

Représentation du sens abstrait

Les représentations AMR des énoncés sont obtenues par la mise en œuvre d'un algorithme d'analyse syntaxico-sémantique. Ces catégories de représentations peuvent être associées à des banques de données annotées complètes, ce qui est le cas pour l'AMR. Des corpus d'entraînement sont ainsi constitués pour des algorithmes mettant en œuvre des techniques computationnelles performantes, basées par exemple sur des modèles neuronaux. De nombreux algorithmes ont ainsi été proposés ces dernières années, comme discuté dans le chapitre 3. Ceux-ci progressent régulièrement, du fait de la disponibilité de données plus complètes et de la mise en œuvre de techniques plus efficaces. La représentation de la figure 9.2 a été obtenue en intégrant l'algorithme STOG [49], l'un des plus performants à ce jour. Les données d'entraînement utilisées proviennent du corpus LDC2017T10 [1], développé par le consortium LDC [2].

La construction de représentations AMR n'est pas une fin en soi, mais un moyen pour aboutir à la définition formelle attendue. L'autre étape importante du processus vise la transformation des structures obtenues dans une démarche contrôlée.

1. https ://catalog.ldc.upenn.edu/LDC2017T10.
2. LDC : Linguistic Data Consortium

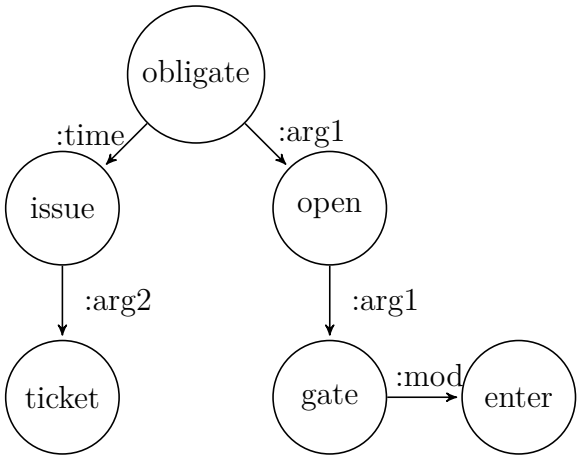


FIGURE 9.2 – Représentation AMR de l'énoncé E_1.

Modalité, propriété et contexte

À ce stade, le problème se pose en termes différents. L'objet du traitement n'est plus un énoncé en langue naturel, mais une structure sémantique. L'étape suivante consiste à interpréter et transformer cette structure pour obtenir la formalisation voulue.

La cible est une relation formelle entre un contexte et une propriété associée à une modalité. Nous proposons un format *ad hoc* précis pour cette relation, que nous nommons MPC[3]. Un élément du langage MPC définit un triplet <*modalité, propriété, contexte*> où :

— la *modalité* spécifie la modalité déontique associée à la propriété, en termes d'obligation, d'interdiction et de possibilité ;

— la *propriété* est une proposition caractérisant un événement ou une propriété sur laquelle s'applique la règle ;

— le *contexte* est une formule de logique propositionnelle spécifiant les conditions d'application de la règle.

Le triplet <*impossible, issue-ticket, (full-parking* $\vee$ *pass-vehicle)*> définit une règle interdisant l'émission d'un ticket si le parking est plein, ou si un véhicule passe. Comme nous le verrons dans la suite du traitement, un triplet MPC peut être dérivé pour construire une spécification formelle vérifiable. L'usage du format MPC n'est pas fondamental, mais contribue à améliorer la lisibilité et la modularité du traitement.

3. MPC : Modality Property Context

Pour obtenir ces triplets, nous proposons un algorithme d'analyse de graphes. Cet algorithme met en œuvre différents principes détaillés dans le chapitre 4. Nous en reprenons les grandes lignes, et renvoyons à la lecture de ce chapitre pour les définitions précises.

Concrètement, le graphe AMR peut être projeté pour lui donner une forme élémentaire, celle d'un graphe sémantique où les nœuds et les arcs sont associés à des étiquettes (figure 9.3). Il est donc possible de s'affranchir de la connaissance précise de ce formalisme pour la suite du traitement, notamment du référentiel des propositions sémantiques (PropBank), même si ces propositions peuvent se retrouver dans les règles de transformation définies. De même, les principes exposés dans la suite restent valables pour une application avec une autre structure intermédiaire, à condition que celle-ci puisse également se projeter sous la forme d'un graphe étiqueté.

Nous introduisons de plus un nouvel élément pour contrôler l'interprétation du graphe sémantique, que nous nommons filet sémantique. Un filet sémantique peut être vu comme un sous-graphe. Il définit un ensemble de sommets, ensemble associé à un type et à une valeur. Nous considérons un ensemble de filets sémantiques, en prenant comme point de départ l'ensemble des filets unitaires pour chaque sommet. Cet ensemble est nommé interprétation sémantique. La figure 9.3 montre l'interprétation initiale du graphe obtenu à l'issue de la phase précédente.

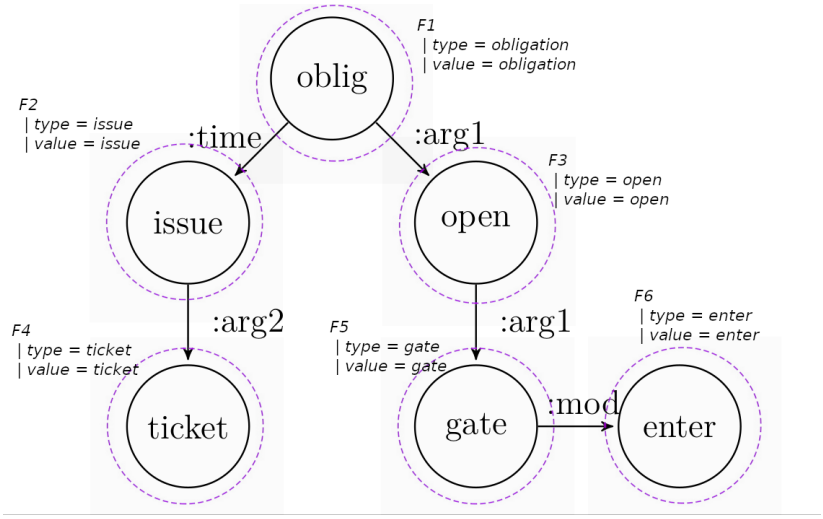


FIGURE 9.3 – Interprétation initiale du graphe sémantique G_1, correspondant à l'énoncé E_1. Ce graphe est une simple projection de la représentation AMR. Les filets sémantiques apparaissent en pointillés. L'interprétation initiale associe un filet à chaque sommet.

La suite consiste à enrichir cet ensemble de filets sémantiques avec de nouveaux fi-

lets calculés à partir des filets déjà définis. Des schémas de calcul sont définis dans ce but. Chaque schéma associe une requête logique et un opérateur. La requête précise les conditions d'application du calcul, tandis que l'opérateur produit les nouveaux filets sémantiques.

Les premières règles permettent d'ajuster les types de chaque filet, ce que l'on peut observer, sur la figure 9.4, pour les filets F_2 et F_4. Le traitement est ensuite itératif. Les nouveaux filets sont construits par composition. La figure 9.4 montre ainsi le calcul d'un nouveau filet (F_{24}) à partir des filets F_2 et F_4 de l'interprétation initiale.

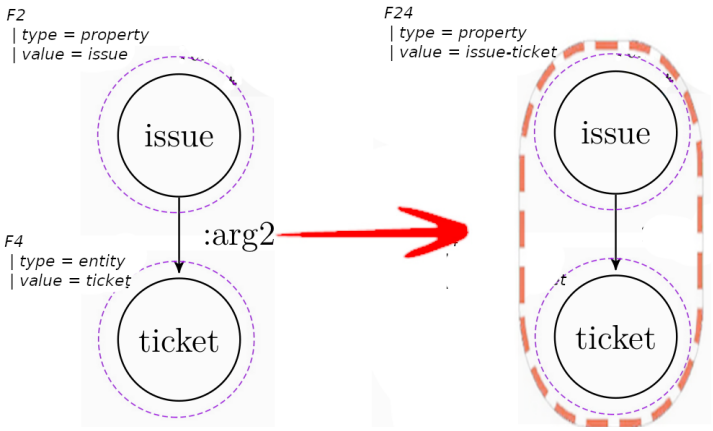


FIGURE 9.4 – Le filet F_{24} est obtenu par composition des filets F_2 et F_4. L'opération appliquée permet d'obtenir un nouvel ensemble de sommets, associé à un nouveau type et une nouvelle valeur, en fonction des types et valeurs de F_2 et F_4.

Finalement, la figure 9.5 montre le calcul itératif sur le graphe initial, qui permet d'obtenir un filet sémantique global associé au type *output*.

Spécification formelle

L'étape précédente a permis d'extraire toutes les exigences comportementales de la spécification étudiée. Précisément, les exigences extraites et formalisées sont celles qui définissent une relation entre un contexte, exprimable sous la forme d'une formule logique propositionnelle, et une propriété associée à une modalité. Ces exigences sont transposées dans un format *ad hoc* facilement interprétable, le format MPC. Le tableau 9.2 regroupe les formalisations obtenues pour les quatre exigences analysées.

La dernière étape du traitement est la composition de ces exigences pour obtenir une spécification formelle en s'appuyant sur une théorie respectant les principes présentés dans

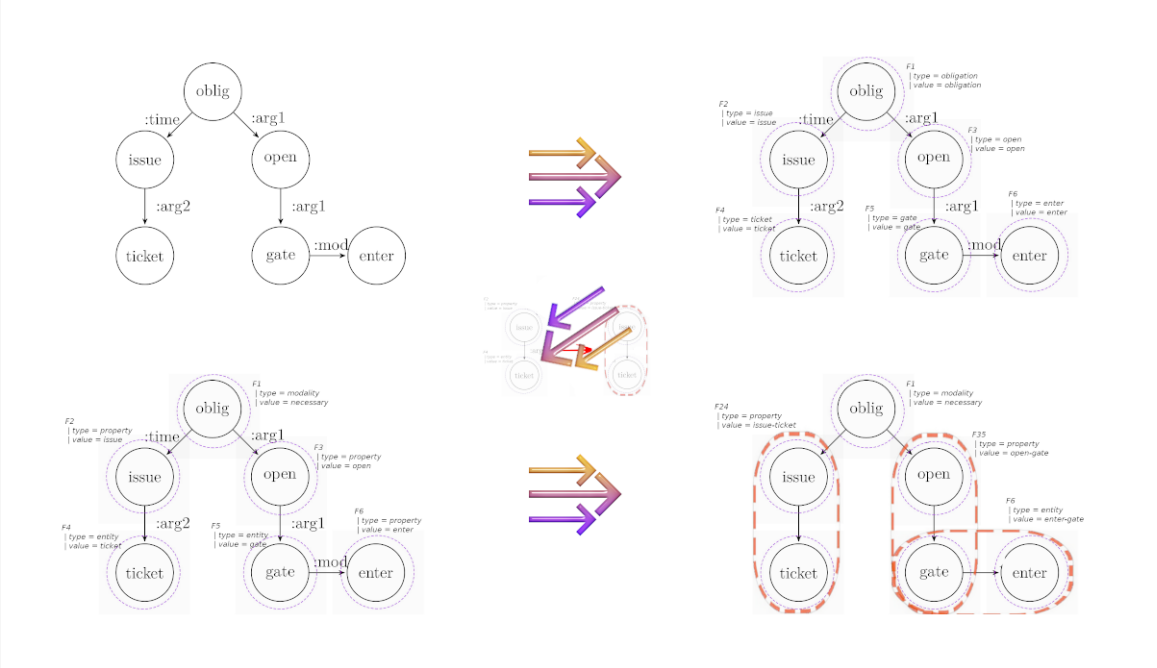


FIGURE 9.5 – Analyse par application des STC du graphe G_1

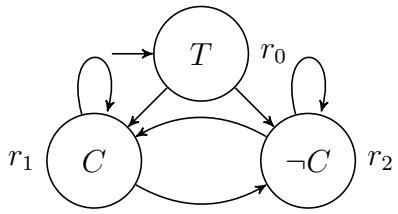
Id	*Exigences formelles (Format MPC)*
E_1	<necessary, open-gate, issue-ticket>
E_2	<impossible, pass-car, close-gate>
E_3	<necessary, open-gate, after insert-ticket>
E_4	<necessary, not open-gate, full-parking or pass-vehicle>

TABLE 9.2 – Formalisation des exigences du tableau 9.1

[4]. Les automates moniteurs et les DPAA sont deux formalismes qui permettent de saisir des exigences comportementales, et notamment les comportements attendus lorsque le système est dans un état particulier. Une théorie d'interface, reposant sur la définition d'une relation de raffinement et d'opérateurs algébriques (composition, produit et quotient), peut être établie, partiellement pour les automates moniteurs, et complètement pour les DPAA.

Plusieurs méthodes peuvent être proposées pour obtenir ces structures en partant des exigences exprimées au format MPC. La définition d'une grammaire complète pour le format MPC, et l'application des techniques de compilation habituelles, est l'approche la plus rigoureuse et complète. Une autre méthode, simple, consiste à définir des patrons pour les différents types de règles rencontrées. Un automate caractéristique peut être associé

à chaque type de règle, comme illustré dans la figure 9.6. Le contexte est directement associé à un état de l'automate, tandis qu'une propriété peut être caractérisée à l'aide des états interdits (pour les automates moniteurs) ou d'ensembles d'acceptation spécifiques (pour les DPAA), en fonction de la modalité associée.



	$Acc(r_i)$
r_0	$2^{2^{2^\Pi}}$
r_1	$\mathcal{A} = f(M, P)$
r_2	$2^{2^{2^\Pi}}$

FIGURE 9.6 – Automate caractéristique pour une règle simple, associant une modalité M, une propriété P et un contexte C. Le tableau donne les ensembles d'acceptation associés à chaque état. L'ensemble associé à l'état r_1 est calculé en fonction de M et P.

Notons que plusieurs automates caractéristiques peuvent être définis pour couvrir l'hétérogénéité des règles. En particulier, des structures dissemblables permettent de traduire des enchaînements d'événements.

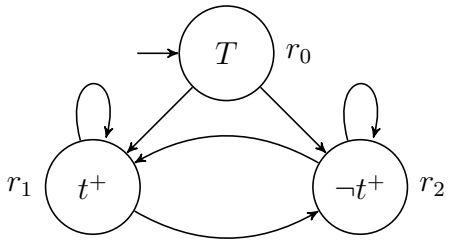
Nous pouvons appliquer cette construction aux exigences du tableau 9.2. Dans un premier temps, l'alphabet des propositions atomiques peut être constitué en prenant toutes les propositions qui apparaissent dans les exigences. Ces propositions sont regroupées dans le tableau 9.3. Un identifiant est utilisé pour simplifier les notations. L'alphabet des propositions est noté $\Pi = \{o, c, t^+, t^-, p, f\}$.

Id	*Exigences formelles (Format MPC)*
o	open-gate
c	close-gate
t^+	issue-ticket
t^-	insert-ticket
p	pass-car | pass-vehicle
f	full-parking

TABLE 9.3 – Ensemble des propositions contenues dans les exigences

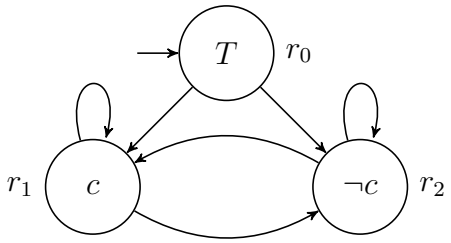
La figure 9.7 traduit l'exigence E_1. La formule t^+ définit le contexte. Elle est associée à l'état r_1 et sa négation à l'état r_2. L'état r_1 est également associé à un ensemble d'acceptation qui implique l'ouverture de la barrière, dans toutes les réalisations compatibles avec la spécification.

Les exigences E_2 à E_4 du tableau 9.2 sont traduites sur le même schéma pour obtenir les automates des figures suivantes (figures 9.8 à 9.10).



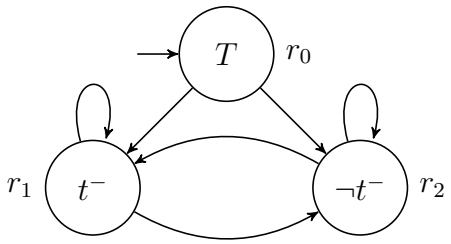
	$Acc(r_i)$
r_0	$2^{2^{2^{\Pi}}}$
r_1	$\{\alpha \mid \forall \sigma \in \alpha, o \in \sigma\}$
r_2	$2^{2^{2^{\Pi}}}$

FIGURE 9.7 – L'automate A_1 est la spécification formelle de l'exigence E_1 (<necessary, open-gate, issue-ticket>).



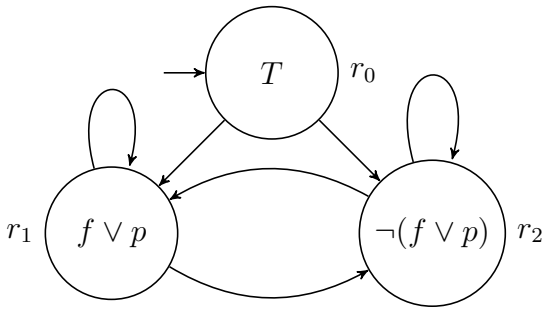
	$Acc(r_i)$
r_0	$2^{2^{2^{\Pi}}}$
r_1	$\{\alpha \mid \forall \sigma \in \alpha, p \notin \sigma\}$
r_2	$2^{2^{2^{\Pi}}}$

FIGURE 9.8 – L'automate A_2 est la spécification formelle de l'exigence E_2 (<impossible, pass-car, close-gate>).



	$Acc(r_i)$
r_0	$2^{2^{2^{\Pi}}}$
r_1	$\{\alpha \mid \forall \sigma \in \alpha, o \in \sigma\}$
r_2	$2^{2^{2^{\Pi}}}$

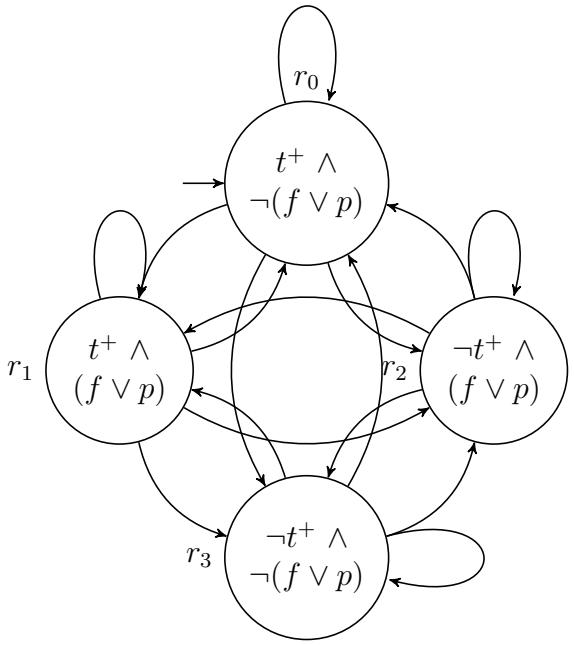
FIGURE 9.9 – L'automate A_3 est la spécification formelle de l'exigence E_3 (<necessary, open-gate, after insert-ticket>).



	$Acc(r_i)$
r_0	$2^{2^{2^{\Pi}}}$
r_1	$\{\alpha \mid \forall \sigma \in \alpha, o \notin \sigma\}$
r_2	$2^{2^{2^{\Pi}}}$

FIGURE 9.10 – L'automate A_4 est la spécification formelle de l'exigence E_4 (<necessary, not open-gate, full-parking or pass-vehicle>).

Les opérations de composition, définies dans les chapitre 7 et 8, peuvent être appliquées sur ces automates. Par exemple, la composition de A_1 et A_4 donne l'automate de la figure 9.11 (l'état initial n'est pas représenté pour simplification). On observe que l'état r_1 de cet automate est associé à un *ensemble vide*. Cela signifie que toute réalisation autorisant l'émission d'un ticket alors qu'un véhicule passe ou que le parking est plein conduit à une situation incohérente. Une nouvelle exigence devrait être ajoutée pour préciser l'impossibilité d'émettre un ticket si une voiture passe, ou si le parking est plein.



	$Acc(r_i)$
r_0	$\{\alpha \mid \forall \sigma \in \alpha, o \in \sigma\}$
r_1	$\emptyset$
r_2	$\{\alpha \mid \forall \sigma \in \alpha, o \notin \sigma\}$
r_3	$2^{2^{2^{\Pi}}}$

FIGURE 9.11 – Composition des automates A_1 et A_4

9.2 Procédure de manipulation d'un jeu de poupées russes

Ce cas d'étude illustre la conception de systèmes dont les actions s'inscrivent dans une procédure, définie par un ensemble de règles. Précisément, nous considérons ici une procédure simple encadrant la manipulation d'un jeu de poupées russes. Le jeu suivi est composé de quatre poupées de tailles différentes, notées respectivement A, B, C et D. Chaque poupée peut se retrouver dans un état ouvert ou un état fermé, en position libre sur un plateau ou à l'intérieur d'une autre poupée. Schématiquement, trois actions peuvent

être observées : l'ouverture d'une poupée, la fermeture d'une poupée et le placement d'une poupée dans une autre poupée. L'objectif fixé est la construction d'une spécification formelle permettant de décider du respect de cette procédure par une modélisation donnée.

Spécification informelle du système

Les exigences regroupées dans le tableau 9.4 précisent les règles de manipulation des poupées russes. Elles sont exprimées en anglais, sans contrainte particulière sur l'énoncé, même si des formulations simples ont été privilégiées. Ces exigences permettent de caractériser les propriétés d'état du système, et les contraintes d'exécution des actions observables.

Id	*Exigences*
E_1	A doll can be opened, closed, free or placed inside another doll.
E_2	If a doll is open, then this doll is not closed.
E_3	A closed doll can be opened. An open doll can be closed.
E_4	Free dolls are placed on a tray.
E_5	A doll is free if it is not inside another doll.
E_6	A free doll can be placed inside another doll.
E_7	A doll placed inside another doll can be removed from this doll.
E_8	A doll cannot be placed inside a closed doll.
E_9	Each doll can receive smaller doll.
E_{10}	A doll can receive a second doll if it is empty.
E_{11}	A doll cannot be closed if it is inside another doll.
E_{12}	A doll that receives another doll is no longer empty.

TABLE 9.4 – Quelques règles de manipulation de poupées russes

Au début du jeu, la taille relative des poupées les unes par rapport aux autres est connue. Toutes les poupées sont fermées, et aucune poupée n'est à l'intérieur d'une autre. Le but du jeu est qu'il ne reste qu'une poupée libre et fermée (les autres poupées étant à l'intérieur de cette poupée ou d'une poupée plus petite).

Représentation pivot et transduction sémantique

La première phase d'analyse permet d'obtenir une première représentation formelle de chaque énoncé, sous la forme de graphe AMR. La figure 9.12 montre en exemple deux graphes AMR, issus de l'analyse des exigences E_3 et E_8.

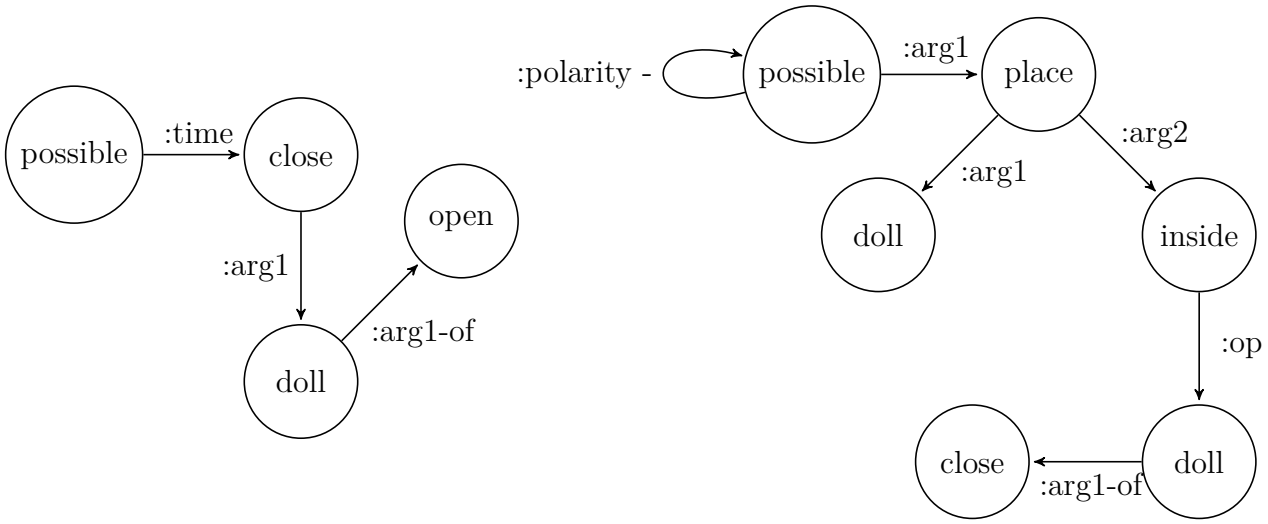


FIGURE 9.12 – Graphes AMR des exigences E_2 et E_5.

La suite du processus met en œuvre un traitement d'analyse par transduction sémantique. Le but est d'extraire les éléments nécessaires pour définir les automates à ensembles d'acceptation correspondant à chacune des exigences. L'alphabet des propositions atomiques est constitué en incluant les propriétés d'états et les événements (vus comme propriétés d'action). Les ensembles d'acceptation sont déterminés en fonction des modalités, dont l'application aux propriétés couvertes se traduit par différentes variabilités de réalisations possibles. L'analyse de la structure générale du graphe permet finalement de distinguer la structure de l'automate. Nous proposons d'utiliser trois attributs distincts pour le calcul de l'alphabet, des ensembles d'acceptation et de l'automate caractéristique, notés respectivement Π, *Acc* et A. Les attributs *prop* et *mod* sont également utilisés.

Soit $\Sigma = \Sigma_P \cup \Sigma_R$ un alphabet de propositions et de relations, et $\mathcal{G} = (S, A, \kappa, \eta)$ un graphe sémantique issu de la représentation AMR, où S est un ensemble de sommets, $A \subseteq S \times S$ est un ensemble d'arcs, $\kappa \subseteq S \times \Sigma_P$ est un ensemble de propositions sur les sommets et $\eta \subseteq A \times \Sigma_R$ est un ensemble de relations sur les arcs.

Dans un premier temps, un filet sémantique est associé à chaque nœud du graphe :

— $\forall s \in S, \mathfrak{f}_s = (s, s, init, \kappa(s))$.

Les types de base associés aux filets découlent de l'analyse du concept associé aux nœuds. L'alphabet des propositions Σ_P est partagé en plusieurs ensembles de concepts, notés $C_{modality}$, C_{entity} et $C_{property}$. Nous distinguons ainsi trois types de base pour différencier les modalités, les entités et les propriétés. Soit s sommet du graphe, le filet $\mathfrak{f}'_s = (s, s, t', \kappa(s)$ est défini avec t' égal à :

— modality si $\kappa(s) \in C_{modality}$,

— entity si $\kappa(s) \in C_{entity}$,

— property si $\kappa(s) \in C_{property}$.

L'ensemble $C_{property}$ est composé des concepts correspondant aux propositions verbales de la PropBank. La définition de la propriété peut être affinée en s'appuyant sur la structure du graphe. Quelques requêtes logiques sont définies pour capturer les filets permettant d'affiner les propriétés. Elle sont associées à un opérateur permettant d'enrichir l'alphabet Π. L'opération $p(\mathfrak{f}, \mathfrak{f}')$ définit une nouvelle propriété (nouvelle valeur pour l'attribut *prop*) à partir des attributs *prop* des filets $\mathfrak{f}$ et $\mathfrak{f}'$. On peut ainsi définir une série de STC tel que $s_{prop} = (req_1, tr_1)$, avec :

— $req_1[\mathfrak{f}, \mathfrak{f}'] = property(\mathfrak{f}) \wedge entity(\mathfrak{f}') \wedge arg1(\mathfrak{f}, \mathfrak{f}')$;

— $tr_1(\mathfrak{f}, \mathfrak{f}')$ est un opérateur de transduction qui permet de définit un filet de type *prop* avec $\Pi = \mathfrak{f}.\Pi \cup \mathfrak{f}'.\Pi \cup p(\mathfrak{f}, \mathfrak{f}')$.

Cet ensemble pourra être nettoyé en post-traitement pour ne garder que les propriétés pertinentes, c'est à dire les propriétés les plus précises.

La relation entre une modalité et une propriété est saisie avec le schéma de transduction $s_{acc} = (req_2, tr_2)$, avec :

— $req_2[\mathfrak{f}, \mathfrak{f}'] = modality(\mathfrak{f}) \wedge property(\mathfrak{f}')$;

— $tr_2(\mathfrak{f}, \mathfrak{f}')$ est un opérateur de transduction qui permet de définit un filet de type *realisation* avec un ensemble d'acceptation *Acc* en fonction de la propriété et de la modalité.

Finalement, plusieurs schémas de transduction peuvent être définis pour déterminer l'automate caractéristique en fonction de la structure du graphe. Les filets de type *realisation* précisent les ensembles d'acceptation associés aux états de l'automate.

L'application de ces opérations fournit l'alphabet propositionnel et les automates à ensembles d'acceptation correspondant aux énoncés. En complément, quelques opérations peuvent être utiles, en cours de traitement et en post-traitement, pour ajuster le résultat obtenu.

Spécification formelle

L'extraction des entités et des propriétés permet de constituer un ensemble de propositions. Les événements observés et les propriétés d'état du système sont définis sous la forme de variables propositionnelles. Soit Π cet alphabet. Les poupées sont notées a, b, c et d, et les différents états possibles des poupées sont extraites de l'exigence E_1. Elles sont traduites par les propriétés $op(x)$, $cl(x)$, $free(x)$, $tray(x)$, $in(x, y)$, $empty(x)$ et $(x < y)$. Les propriétés $p(x, y)$ et $r(x, y)$ traduisent les actions d'insertion et de retrait d'une poupée (par rapport à une autre poupée), tandis que les événements $op(x)$ et $cl(x)$ peuvent être, pour simplification, confondus avec les propriétés d'états qui en découlent. Cet alphabet est décrit dans le tableau 9.5.

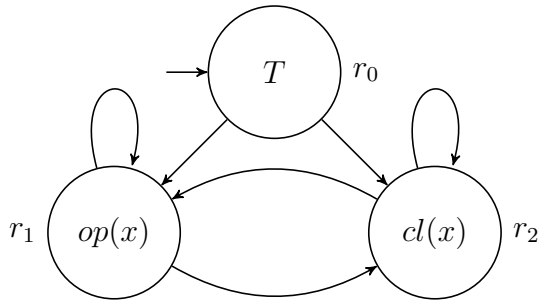
Id	A/P	*Propriété (fr)*	*Property (en)*
$free(x)$	P	Poupée libre	Free doll
$tray(x)$	P	Poupée sur le plateau	Free on a tray
$in(x, y)$	P	Poupée x placée dans y	Doll x in doll y
$empty(x)$	P	Poupée x vide	Empty doll
$(x < y)$	P	La poupée x est plus petite que y	Doll x smaller than y
$op(x)$	A/P	Ouverture d'une poupée	Open doll
$cl(x)$	A/P	Fermeture d'une poupée	Close doll
$p(x, y)$	A	Insertion de la poupée x dans y	Place x inside y
$r(x, y)$	A	Retrait de la poupée x	Remove x from y

TABLE 9.5 – Alphabet des actions observés (A) et des propriétés d'état (P)

L'analyse par transduction sémantique est appliquée pour produire les automates traduisant les exigences du document de spécification. Ces constructions peuvent être calculées directement en suivant les indications de la section 9.2. L'application récursive des règles de transduction permet ainsi de déduire les ensembles d'acceptation et les structures d'automate adéquates. Il est également possible d'obtenir les représentations voulues indirectement, en passant par le format MPC (approche utilisée sur l'exemple du parking, section 9.1). Dans les deux cas, les exigences analysées sont traduites sous la forme d'automates d'acceptation propositionnels (DPAA) qu'il est possible de combiner.

La figure 9.13 montre l'automate caractéristique des exigences E_2 et E_3. L'exigence E_2 se reflète dans les ensembles d'acceptation associés à l'état r_0, tandis que les états r_1 et r_2 traduisent les deux règles de l'exigence E_3.

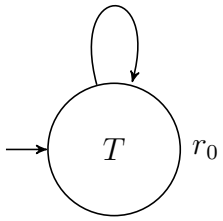
La prise en compte des exigences E_4, E_5 et E_9 autorise le raffinement des ensembles d'acceptation envisageables. Ces ensembles peuvent être associés à un automate élémen-



	$Acc(r_i)$
r_0	$\{\alpha \mid \forall\sigma \in \alpha,\ (op(x) \wedge cl(x))\}$
r_1	$\{\alpha \mid \exists\sigma \in \alpha,\ cl(x) \in \sigma\}$
r_2	$\{\alpha \mid \exists\sigma \in \alpha,\ op(x) \in \sigma\}$

FIGURE 9.13 – L'automate A_1 est la spécification formelle des exigences E_2 et E_3.

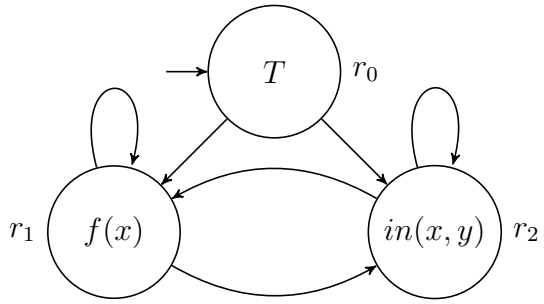
taire ne comportant qu'un unique état associé à la formule $True$ (figure 9.14).



	$Acc(r_i)$
r_0	$\{\alpha \mid \forall\sigma \in \alpha,$ $(\forall x, op(x) \in \sigma \Leftrightarrow cl(x) \notin \sigma),$ $(free(x) \in \sigma \Rightarrow tray(x) \in \sigma),$ $(free(x) \in \sigma \Leftrightarrow \exists y, in(x,y) \in \sigma),$ $(\exists y, in(y,x) \in \sigma \Rightarrow empty(x) \notin \sigma)\ \}$

FIGURE 9.14 – L'automate A_2 modélise les exigences E_4, E_5 et E_9.

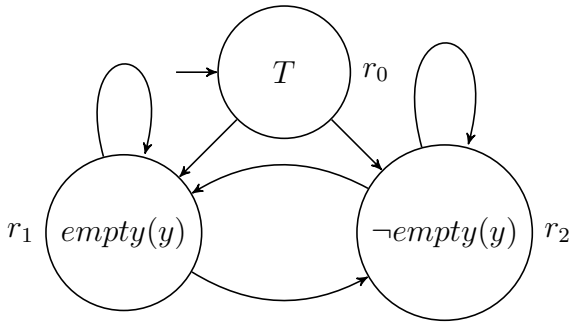
L'automate de la figure 9.15 modélise les exigences E_6 et E_7.



	$Acc(r_i)$
r_0	$\{\alpha \mid \forall\sigma \in \alpha,\ (op(x) \wedge cl(x))\}$
r_1	$\{\alpha \mid \exists\sigma \in \alpha,\ p(x,y) \in \sigma\}$
r_2	$\{\alpha \mid \exists\sigma \in \alpha,\ r(x,y) \in \sigma\}$

FIGURE 9.15 – L'automate A_3 modélise les exigences E_6 et E_7.

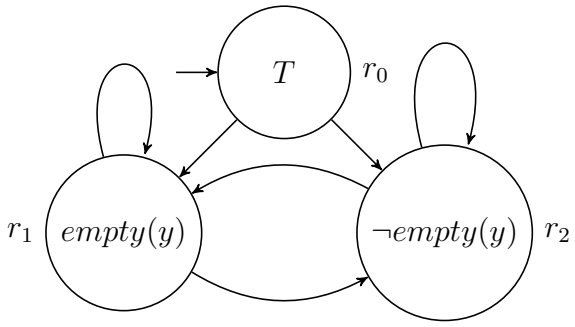
Des structures similaires sont exploitées pour les exigences $E_1 0$, $E_1 1$ et $E_1 2$, qui concernent l'insertion d'une poupée russe dans une autre poupée et l'impossibilité de fermeture d'une poupée (si elle est dans une autre poupée). La figure 9.16 montre la structure obtenue pour l'exigence $E_1 0$, qui définit la possibilité d'insertion d'une poupée dans une autre poupée vide. Notons que cette exigence est ambiguë, car elle n'interdit pas formellement l'insertion d'une poupée dans une poupée vide. Une exigence supplémentaire est nécessaire pour lever l'ambiguïté.



	$Acc(r_i)$
r_0	$2^{2^{2^{\Pi}}}$
r_1	$\{\alpha \mid \exists \sigma \in \alpha,\, in(x,y) \in \sigma\}$
r_2	$2^{2^{2^{\Pi}}}$

FIGURE 9.16 – L'automate A_4 modélise l'exigence $E_1 0$.

Une nouvelle exigence $E_1'0$ peut être introduite pour compléter cette règle. L'énoncé "A doll cannot receive a second doll if it is not empty" peut ainsi être traité pour affiner la représentation (figure 9.17).



	$Acc(r_i)$
r_0	$2^{2^{2^{\Pi}}}$
r_1	$\{\alpha \mid \exists \sigma \in \alpha,\, in(x,y) \in \sigma\}$
r_2	$\{\alpha \mid \forall \sigma \in \alpha,\, in(x,y) \notin \sigma\}$

FIGURE 9.17 – L'automate A_4' modélise les exigences $E_1 0$ et $E_1'0$.

La modélisation complète s'obtient par composition des différents automates, en appliquant les définitions des chapitres 6 et 8. La construction effective d'un automate complet permet de démontrer l'absence d'états inconsistants (c'est à dire l'absence d'états associés à un ensemble inconsistant). Plusieurs réalisations peuvent être envisagées, dont il est possible de démontrer la conformité avec la spécification.

Finalement, la figure 9.18 montre un modèle de Kripke représentant l'évolution d'un modèle pour un système manipulant deux poupées russes, dont les états sont associés aux ensembles de propositions du tableau 9.6. On observe que cette réalisation permet de résoudre le défi demandé, à savoir l'obtention d'une seule poupée libre contenant toutes les autres poupées (une seule poupée dans l'exemple de la figure).

Le tableau 9.6 spécifie les ensembles de propositions validées pour chacun des états du modèle de Kripke donné par la figure 9.18. Le même modèle peut être étendu pour un système manipulant quatre poupées russes (et plus).

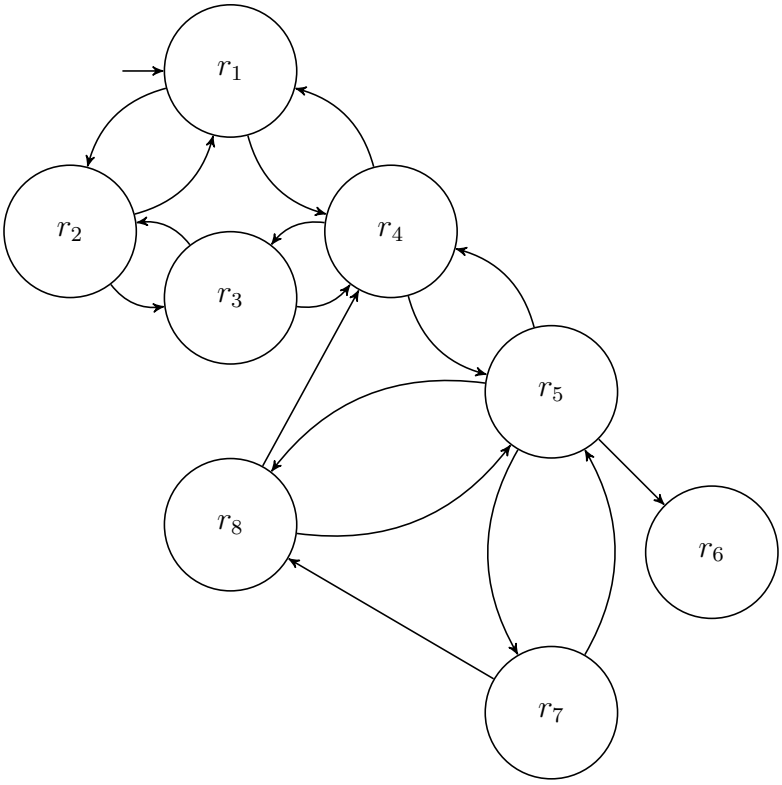


FIGURE 9.18 – Modèle d'un système manipulant deux poupées russes, conforme aux spécifications.

État	*Propositions valides*
r_1	$cl(1), cl(2), free(1), tray(1), free(2), tray(2),$ $empty(1), empty(2), 1 < 2$
r_2	$op(1), cl(2), free(1), tray(1), free(2), tray(2),$ $empty(1), empty(2), 1 < 2$
r_3	$op(1), op(2), free(1), tray(1), free(2), tray(2),$ $empty(1), empty(2), 1 < 2$
r_4	$cl(1), op(2), free(1), tray(1), free(2), tray(2),$ $empty(1), empty(2), 1 < 2$
r_5	$cl(1), op(2), free(2), tray(2),$ $p(1,2), empty(1), 1 < 2, in(1,2)$
r_6	$cl(1), cl(2), free(2), tray(2),$ $empty(1), 1 < 2, in(1,2)$
r_7	$cl(1), op(2), free(2), tray(2),$ $empty(1), 1 < 2, in(1,2)$
r_8	$cl(1), op(2), free(1), tray(1), free(2), tray(2),$ $r(1,2), empty(1), empty(2), 1 < 2$

TABLE 9.6 – Ensembles des propositions valides associées au modèle de Kripke de la figure 9.18

Chapitre 10

Conclusion

Le traitement et la compréhension d'une langue naturelle par des processus autonomes révèlent des enjeux de plus en plus importants. Ceux-ci se reflètent en partie dans le domaine de l'ingénierie des exigences, comme montré par la démarche rapportée dans ce mémoire.

Ainsi, l'application de méthodes formelles pour assister la conception de systèmes s'appuie sur une modélisation des comportements attendus. L'élaboration de ces modèles nécessite d'exprimer les propriétés à vérifier, caractérisées sous la forme d'exigences. Elles peuvent, par exemple, définir les invariants attendus d'un composant, ou les contraintes requises pour une action donnée. Ces exigences se traduisent difficilement dans les langages formels, même si elles y apparaissent de manière précise. *A contrario*, les langues naturelles apportent plus de souplesses, mais également ambiguïtés et imprécisions.

Nous avons cherché, dans nos travaux, à définir les bases d'un processus automatique pour l'analyse de documents de spécification. Notre contribution intègre deux volets complémentaires : le premier permet l'extraction et la formalisation des exigences à partir d'énoncés en langage naturel, le second définit un cadre de raisonnement formel pour accompagner la conception de systèmes.

Pour parvenir à la compréhension des langues naturelles par des procédures automatiques, nous défendons l'usage de représentations structurées ayant les ingrédients nécessaires à la saisie du sens d'un texte. Leurs développements répondent à des objectifs pratiques variés, tels que l'analyse ou la synthèse de textes. Notre proposition met à profit ce type de représentations pour l'étude de documents techniques.

D'autre part, les banques de données associées aux structures sémantiques forment des ressources précieuses pour l'entrainement d'algorithmes performants, dont la conception révèle un champs de recherche actif. Les principes de transduction sémantique [6], exposés dans le chapitre 4, complètent cette démarche en tirant parti de formalismes existants pour des tâches d'analyse et de formalisation d'énoncés. Le cadre proposé s'applique à toute représentation à base de graphes, telle que les graphes AMR ou les structures UCCA.

Ces principes apparaissent comme une alternative aux techniques basées sur des règles de réécriture de graphes ou d'analyse syntaxique. Il s'agit, en quelque sorte, d'une extension introduisant une certaine souplesse dans l'interprétation structurelle et l'application des opérations réalisées. De fait, le passage de la structure pivot à la représentation cible pourrait être envisagé en s'appuyant sur des règles de grammaire et la construction d'un arbre de syntaxe abstraite. Notre approche est similaire. La différence principale réside dans l'introduction d'une plus grande flexibilité dans l'interprétation de la structure sémantique avec l'usage des requêtes logiques. La conséquence intervient sur le contrôle des calculs réalisés. Cette spécificité s'est notamment traduite par la définition de deux concepts clés, les filets sémantiques et les schémas de transduction sémantique. Ainsi, les filets sémantiques produisent naturellement différents ensembles concurrents, traduisant des interprétations et calculs alternatifs. Le typage associé autorise la maîtrise de ces alternatives, pour finalement retenir l'interprétation recherchée, c'est à dire celle qui permet de construire la représentation voulue.

L'exploration de modèles formels appropriés pour soutenir la conception de systèmes réactifs est l'autre orientation explorée dans ce mémoire. Dans ce contexte, le raisonnement par contrat offre un cadre formel efficace et générique [4]. Cette approche définit en effet une théorie supportant différents formalismes et processus de conception. C'est le cas des théories d'interfaces, telles que les automates d'interface [3] et les interfaces modales [13, 14]. Celles-ci fournissent une spécification fusionnée du comportement attendu de composants en cours de conception et des environnements possibles dans lesquels ceux-ci peuvent opérer. Ces approches ont en commun une algèbre de composition et un concept de raffinement, reflétant la décomposition des exigences au niveau du système selon plusieurs composants.

Nos propositions s'inscrivent pleinement dans ce cadre théorique. Les automates déterministes à ensembles d'acceptation propositionnels (DPAA) ont ainsi été proposés pour saisir les exigences de systèmes réactifs en temps discret [8]. Ce formalisme de spécification offre un bon compromis en termes d'expressivité, de propriétés algébriques et de complexité algorithmique. Sa facilité d'utilisation le rend accessible à des ingénieurs non informaticiens, méconnaissant les logiques temporelles. Précisément, les DPAA combinent des propriétés d'état, exprimées sous forme de formules propositionnelles, et des propriétés temporelles simples en temps discret, exprimées sous forme d'actions obligatoires et interdites chaque fois qu'une propriété d'état donnée est présente. Ils étendent les systèmes de transition modale à un cadre propositionnel, où les modèles sont des structures

de Kripke, plutôt que des systèmes de transition d'états étiquetés. Les opérateurs de composition sont fournis. Une théorie d'interface complète est obtenue par suite, avec une relation de raffinement, des opérateurs de composition parallèle, de conjonction et de quotient. Une représentation implicite utilisant des fonctions caractéristiques permet de limiter la complexité de calcul dans le temps et l'espace.

La validation de la méthodologie proposée passe par une mise en œuvre concrète, ce que nous avons réalisé en partie. Il serait intéressant d'étendre nos développements pour une application et une évaluation rigoureuse sur des systèmes réels. La généralisation des DPAA pour prendre en compte les propriétés numériques des systèmes cyber-physiques représente également une perspective intéressante. Cette orientation consisterait à modifier l'élément de base des ensembles d'acceptation, à savoir la proposition atomique, pour prendre en compte des trajectoires numériques.

BIBLIOGRAPHIE

[1] E. M. Clarke and E. A. Emerson, "Synthesis of synchronization skeletons for branching time temporal logic," in *Logic of programs : Workshop*, 1981, vol. 131, pp. 244–263.

[2] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications." New York, NY, USA : ACM, 1986, vol. 8, no. 2, pp. 244–263. [Online]. Available : http://doi.acm.org/10.1145/5397.5399

[3] L. de Alfaro and T. A. Henzinger, "Interface automata," *SIGSOFT Softw. Eng. Notes*, vol. 26, no. 5, pp. 109–120, September 2001.

[4] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger, and K. G. Larsen, *Contracts for System Design*, 2018. [Online]. Available : http://dx.doi.org/10.1561/1000000053

[5] A. Lamercerie, "ARES : un extracteur d'exigences pour la modélisation de systèmes," in *Extraction et Gestion des Connaissances (EGC), Workshop TextMine*, Bruxelles, France, Jan 2020. [Online]. Available : https://hal.archives-ouvertes.fr/hal-02971727

[6] ——, "Transduction sémantique pour la modélisation de système," in *Plate-Forme de l'Intelligence Artificielle (PFIA), rencontres RJCIA*, Angers, France, Jun 2020. [Online]. Available : https://hal.archives-ouvertes.fr/hal-02971742

[7] ——, "Une algèbre des automates d'acceptation propositionnelle déterministes comme théorie d'interface pour la conception de systèmes cyberphysiques," 2019, poster. [Online]. Available : https://hal.archives-ouvertes.fr/hal-02432696

[8] A. Lamercerie and B. Caillaud, "An Algebra of Deterministic Propositional Acceptance Automata (DPAA)," in *Forum on specification & Design Languages (FDL)*, Kiel, Germany, Sep 2020. [Online]. Available : https://hal.archives-ouvertes.fr/hal-02971772

[9] N. Kekatos, “Formal verification of cyber-physical systems in the industrial model-based design process,” Ph.D. dissertation, Université Grenoble Alpes, 2018.

[10] N. Halbwachs, F. Lagnier, and P. Raymond, “Synchronous observers and the verification of reactive systems,” in *Algebraic Methodology and Software Technology (AMAST’93)*, M. Nivat, C. Rattray, T. Rus, and G. Scollo, Eds. Springer London, 1994, pp. 83–96.

[11] D. Parnas and P. Clements, “A rational design process : How and why to fake it.” *Software Engineering, IEEE Transactions on*, vol. SE-12, pp. 251–257, 02 1986.

[12] C. Heitmeyer, R. Jeffords, and B. Labaw, “Automated consistency checking of requirements specifications,” *ACM Trans. Softw. Eng. Methodol.*, vol. 5, pp. 231–261, 1996.

[13] J. Raclet, E. Badouel, A. Benveniste, B. Caillaud, A. Legay, and R. Passerone, “A modal interface theory for component-based design,” *Fundamenta Informaticae*, vol. 108, no. 1-2, pp. 119–149, 2010.

[14] F. Bujtor, S. Fendrich, G. Lüttgen, and W. Vogler, “Nondeterministic modal interfaces,” in *SOFSEM 2015 : Theory and Practice of Computer Science*. Springer Berlin Heidelberg, 2015, pp. 152–163.

[15] B. Meyer, *Touch of Class : Learning to Program Well Using Object Technology and Design by Contract*. Springer, Software Engineering, 2009.

[16] B. Caillaud, “Mica : A Modal Interface Compositional Analysis Library,” October 2011.

[17] M. Steedman, *The syntactic process*. MIT Press, 2000.

[18] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, “Property specification patterns for finite-state verification,” in *Proceedings of the Second Workshop on Formal Methods in Software Practice*, ser. FMSP ’98. New York, NY, USA : Association for Computing Machinery, 1998, p. 7–15. [Online]. Available : https://doi.org/10.1145/298595.298598

[19] ——, “Patterns in property specifications for finite-state verification,” in *Proceedings of the 21st International Conference on Software Engineering*, ser. ICSE ’99. New York, NY, USA : ACM, 1999, pp. 411–420. [Online]. Available : http://doi.acm.org/10.1145/302405.302672

[20] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*. Berlin, Heidelberg : Springer-Verlag, 1992.

[21] K. M. Olender and L. J. Osterweil, *Cecil : A Sequencing Constraint Language for Automatic Static Analysis Generation.* Berlin, Heidelberg : Springer Berlin Heidelberg, 2011, pp. 115–141. [Online]. Available : https://doi.org/10.1007/978-3-642-19823-6_7

[22] S. Konrad and B. H. C. Cheng, "Real-time specification patterns," in *Proceedings of the 27th International Conference on Software Engineering*, ser. ICSE '05. New York, NY, USA : ACM, 2005, pp. 372–381.

[23] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Systems*, vol. 2, pp. 255–299, 1990. [Online]. Available : https://doi.org/10.1007/BF01995674

[24] R. Alur, "Techniques for automatic verification of real-time systems," Ph.D. dissertation, Stanford University, 1991.

[25] N. E. Fuchs and R. Schwitter, "Specifying logic programs in controlled natural language," *CoRR*, vol. abs/cmp-lg/9507009, 1995.

[26] H. Kamp and U. Reyle, *From Discourse to Logic : Introduction to Model-theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*, ser. Studies in Linguistics and Philosophy. Dordrecht : Springer, 1993, vol. 42.

[27] W3C OWL Working Group, "OWL-2 Web Ontology Language Document Overview (Second Edition) - W3C Recommendation 11 December 2012," World Wide Web Consortium (W3C), 2012. [Online]. Available : http://www.w3.org/TR/owl2-overview/

[28] I. Horrocks, P. Patel-Schneider, H. Boley, S. Tabet, B. Grossof, and M. Dean, "Swrl : A semantic web rule language combining owl and ruleml," *W3C Subm*, vol. 21, 01 2004.

[29] T. Kuhn, "Acerules : Executing rules in controlled natural language," vol. 4524, 06 2007, pp. 299–308.

[30] R. Montague, *Formal Philosophy : Selected Papers of Richard Montague*, ser. A Yale Paperbound. Yale University Press, 1974.

[31] K. Ajdukiewicz, "Die syntaktische Konnexität." Linguistic Society of America, 1935, vol. 1, pp. 1–27, traduction française : [102].

[32] Y. Bar-Hillel, "A quasi-arithmetical notation for syntactic description," *Language*, vol. 29, pp. 47–58, 1953.

[33] J. Lambek, "The mathematics of sentence structure," *American Mathematical Monthly*, vol. 154-170, 1958.

[34] Y. Bar Hillel, C. Gaifman, and E. Shamir, "On Categorial and Phrase Structure Grammars," *Bulletin of the Research Council of Israel*, vol. 9F, 1960.

[35] M. Lewis and M. Steedman, "A* CCG parsing with a supertag-factored model," in *Proceedings of the Conference on Empirical Methods in Natural Language Processin*, 2014.

[36] K. Vijay-Shanker and D. J. Weir, "Parsing some constrained grammar formalisms," *Comput. Linguist.*, vol. 19, no. 4, pp. 591–636, 1993.

[37] J. Bos, "Open-domain semantic parsing with boxer," *Proceedings of the 20th Nordic Conference of Computational Linguistics*, pp. 301–304, 2015.

[38] T. Matsuzaki, T. Ito, H. Iwane, H. Anai, and N. H. Arai, "Semantic parsing of pre-university math problems," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*. Vancouver, Canada : Association for Computational Linguistics, July 2017, pp. 2131–2141. [Online]. Available : https://www.aclweb.org/anthology/P17-1195

[39] P. de Groote, "Towards abstract categorial grammars," in *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, ser. ACL '01. Association for Computational Linguistics, 2001, pp. 252–259.

[40] C. Pradel, O. Haemmerlé, and N. Hernandez, "Natural language query interpretation into SPARQL using patterns," in *Fourth International Workshop on Consuming Linked Data - COLD 2013*, Sydney, Australia, 2013, pp. pp. 1–12. [Online]. Available : https://hal.archives-ouvertes.fr/hal-01143219

[41] D. Sadoun, C. Dubois, Y. Ghamri-Doudane, and B. Grau, "From natural language requirements to formal specification using an ontology," in *IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI 2013)*, Herndon, VA, United States, 2013, pp. 755–760.

[42] H. Kamp, "A theory of truth and semantic representation," in *Formal Methods in the Study of Language*, J. A. G. Groenendijk, T. M. V. Janssen, and M. B. J. Stokhof, Eds. Amsterdam : Mathematisch Centrum, 1981, vol. 1, pp. 277–322.

[43] R. Nelken and N. Francez, "Automatic translation of natural language system specifications into temporal logic," 1996.

[44] S. Ghosh, D. Elenius, W. Li, P. Lincoln, N. Shankar, and W. Steiner, "Arsenal : Automatic requirements specification extraction from natural language," vol. abs/1403.3142, 2014. [Online]. Available : http://arxiv.org/abs/1403.3142

[45] S. Bensalem, V. Ganesh, Y. Lakhnech, C. Muñoz, S. Owre, H. Rueß, J. Rushby, V. Rusu, H. Saïdi, N. Shankar, E. Singerman, and A. Tiwari, "An overview of SAL," in *LFM 2000 : Fifth NASA Langley Formal Methods Workshop*, 2000, pp. 187–196.

[46] L. Banarescu, C. Bonial, S. Cai, M. Georgescu, K. Griffitt, U. Hermjakob, K. Knight, P. Koehn, M. Palmer, and N. Schneider, "Abstract meaning representation for sembanking," in *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*. Sofia, Bulgaria : Association for Computational Linguistics, 2013, pp. 178–186.

[47] O. Abend and A. Rappoport, "Universal conceptual cognitive annotation (UCCA)," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*. Sofia, Bulgaria : Association for Computational Linguistics, Aug 2013, pp. 228–238. [Online]. Available : https://www.aclweb.org/anthology/P13-1023

[48] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics : System Demonstrations*. Baltimore, Maryland : Association for Computational Linguistics, June 2014, pp. 55–60. [Online]. Available : https://www.aclweb.org/anthology/P14-5010

[49] S. Zhang, X. Ma, K. Duh, and B. Van Durme, "AMR Parsing as Sequence-to-Graph Transduction," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy : Association for Computational Linguistics, 2019, pp. 80–94. [Online]. Available : https://www.aclweb.org/anthology/P19-1009

[50] M. Palmer, D. Gildea, and P. Kingsbury, "The proposition bank : An annotated corpus of semantic roles," *Computational Linguistics*, vol. 31, no. 1, pp. 71–106, 2005. [Online]. Available : https://www.aclweb.org/anthology/J05-1004

[51] J. Raclet, "Quotient de spécifications pour la réutilisation de composants," Ph.D. dissertation, Université de Rennes 1, 2007.

[52] K. G. Larsen and B. Thomsen, "A modal process logic," in *Proceedings. Third Annual Symposium on Logic in Computer Science*, 1988, pp. 203–210.

[53] B. Jonsson and K. G. Larsen, "Specification and refinement of probabilistic processes," in *[1991] Proceedings Sixth Annual IEEE Symposium on Logic in Computer Science*, July 1991, pp. 266–277.

[54] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of English : The Penn Treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1993. [Online]. Available : https://www.aclweb.org/anthology/J93-2004

[55] A. Meyers, R. Reeves, C. Macleod, R. Szekely, V. Zielinska, B. Young, and R. Grishman, "The nombank project : An interim report," in *HLT-NAACL 2004 Workshop : Frontiers in Corpus Annotation*, A. Meyers, Ed. Boston, Massachusetts, USA : Association for Computational Linguistics, May 2 - May 7 2004, pp. 24–31.

[56] C. F. Baker, C. J. Fillmore, and J. B. Lowe, "The Berkeley FrameNet Project," in *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, ser. ACL 1998 / COLING 1998, vol. 1. Montreal, Quebec, Canada : Association for Computational Linguistics, 1998, p. 86–90. [Online]. Available : https://doi.org/10.3115/980845.980860

[57] M. Surdeanu, R. Johansson, A. Meyers, L. Màrquez, and J. Nivre, "The CoNLL 2008 shared task on joint parsing of syntactic and semantic dependencies," in *CoNLL 2008 : Proceedings of the Twelfth Conference on Computational Natural Language Learning*. Manchester, England : COLING 2008 Organizing Committee, 2008, pp. 159–177. [Online]. Available : https://www.aclweb.org/anthology/W08-2121

[58] P. Sgall, "Underlying structure of sentences and its relations to semantics," in *Wiener Slawistischer Almanach. Sonderband 33*, T. Reuthe, Ed. Wien : Gesellschaft zur Förderung slawistischer Studien, 1992, pp. 273–282.

[59] A. Böhmová, J. Hajič, E. Hajičová, and B. Hladká, "The prague dependency treebank : Three-level annotation scenario," in *Treebanks : Building and Using Syntactically Annotated Corpora*, A. Abeillé, Ed. Kluwer Academic Publishers, 2001.

[60] X. Pan, T. Cassidy, U. Hermjakob, H. Ji, and K. Knight, "Unsupervised entity linking with Abstract Meaning Representation," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational*

Linguistics : Human Language Technologies. Denver, Colorado : Association for Computational Linguistics, 2015, pp. 1130–1139. [Online]. Available : https://www.aclweb.org/anthology/N15-1119

[61] F. Liu, J. Flanigan, S. Thomson, N. Sadeh, and N. A. Smith, "Toward abstractive summarization using semantic representations," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies*. Denver, Colorado : Association for Computational Linguistics, 2015, pp. 1077–1086. [Online]. Available : https://www.aclweb.org/anthology/N15-1114

[62] S. Garg, A. Galstyan, U. Hermjakob, and D. Marcu, "Extracting Biomolecular Interactions Using Semantic Parsing of Biomedical Texts," *CoRR*, vol. abs/1512.01587, 2015. [Online]. Available : http://arxiv.org/abs/1512.01587

[63] S. Rao, D. Marcu, K. Knight, and H. Daumé III, "Biomedical event extraction using Abstract Meaning Representation," in *BioNLP 2017*. Vancouver, Canada, : Association for Computational Linguistics, Aug 2017, pp. 126–135. [Online]. Available : https://www.aclweb.org/anthology/W17-2315

[64] O. Abend and A. Rappoport, "The state of the art in semantic representation," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, vol. 1. Vancouver, Canada : Association for Computational Linguistics, Jul 2017, pp. 77–89. [Online]. Available : https://www.aclweb.org/anthology/P17-1008

[65] A. S. White, D. Reisinger, K. Sakaguchi, T. Vieira, S. Zhang, R. Rudinger, K. Rawlins, and B. Van Durme, "Universal decompositional semantics on Universal Dependencies," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas : Association for Computational Linguistics, Nov 2016, pp. 1713–1723. [Online]. Available : https://www.aclweb.org/anthology/D16-1177

[66] J. Nivre, M.-C. de Marneffe, F. Ginter, Y. Goldberg, J. Hajič, C. D. Manning, R. McDonald, S. Petrov, S. Pyysalo, N. Silveira, R. Tsarfaty, and D. Zeman, "Universal Dependencies v1 : A multilingual treebank collection," in *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. Portorož, Slovenia : European Language

Resources Association (ELRA), May 2016, pp. 1659–1666. [Online]. Available : https://www.aclweb.org/anthology/L16-1262

[67] K. Droganova and D. Zeman, "Towards deep Universal Dependencies," in *Proceedings of the Fifth International Conference on Dependency Linguistics (Depling, SyntaxFest 2019)*. Paris, France : Association for Computational Linguistics, Aug 2019, pp. 144–152. [Online]. Available : https://www.aclweb.org/anthology/W19-7717

[68] J. Bos, V. Basile, K. Evang, N. Venhuizen, and J. Bjerva, *The Groningen Meaning Bank*, 06 2017, pp. 463–496.

[69] P. Geach, *Reference and Generality : An Examination of Some Medieval and Modern Theories*, ser. Contemporary philosophy. Cornell University Press, 1962. [Online]. Available : https://books.google.fr/books?id=HluvAAAAIAAJ

[70] L. Ahrenberg, A. Jönsson, and N. Dahlbäck, "Discourse representation and discourse management for a natural language dialogue system," in *In Proceedings of the Second Nordic Conference on Text Comprehension in Man and Machine, Taby*, 1990.

[71] S. Lauria, G. Bugmann, T. Kyriacou, J. Bos, and E. Klein, "Training personal robots using natural language instruction," *Intelligent Systems, IEEE*, vol. 16, pp. 38 – 45, 10 2001.

[72] M. Čmejrek, J. Hajič, and V. Kuboň, "Prague czech-english dependency treebank : Any hopes for a common annotation scheme," in *In Proceedings of HLT/NAACL 2004 Workshop : Frontiers in Corpus Annotation*, 2004, pp. 47–54.

[73] P. Sgall, E. Hajičová, and J. Panevová, *The Meaning of the Sentence and Its Semantic and Pragmatic Aspects*. Prague, Czech Republic/Dordrecht, Netherlands : Academia/Reidel Publishing Company, 1986.

[74] P. Sgall, *Generativní popis jazyka a česká deklinace*. Prague, Czech Republic : Academia, 1967.

[75] R. Dixon, *Basic Linguistic Theory Volume 1 : Methodology*, ser. Basic Linguistic Theory. OUP Oxford, 2009. [Online]. Available : https://books.google.fr/books?id=LjHM-iVUjB8C

[76] B. Bianca, B. Laura, B. Claire, C. Shu, K. Philipp, G. Madalina, G. Kira, H. Ulf, K. Kevin, O. Tim, P. Martha, and S. Nathan, "Abstract Meaning Representa-

tion (AMR) 1.2.6 Specification," https://github.com/amrisi/amr-guidelines/blob/master/amr.md, May 2019.

[77] R. Weischedel, E. Hovy, M. Marcus, M. Palmer, R. Belvin, S. Pradhan, L. Ramshaw, and N. Xue, *OntoNotes : A Large Training Corpus for Enhanced Processing*, 01 2011.

[78] S. Cai and K. Knight, "Smatch : an evaluation metric for semantic feature structures," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*. Sofia, Bulgaria : Association for Computational Linguistics, 2013, pp. 748–752.

[79] J. Flanigan, S. Thomson, J. Carbonell, C. Dyer, and N. A. Smith, "A discriminative graph-based parser for the abstract meaning representation," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*. Baltimore, Maryland : Association for Computational Linguistics, 2014, pp. 1426–1436.

[80] J. Zhou, F. Xu, H. Uszkoreit, W. Qu, R. Li, and Y. Gu, "AMR parsing with an incremental joint model," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas : Association for Computational Linguistics, 2016, pp. 680–689. [Online]. Available : https://www.aclweb.org/anthology/D16-1065

[81] Y. Liu, W. Che, B. Zheng, B. Qin, and T. Liu, "An AMR Aligner Tuned by Transition-based Parser," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium : Association for Computational Linguistics, 2018, p. 2422–2430. [Online]. Available : https://www.aclweb.org/anthology/D18-1264

[82] C. Wang, N. Xue, and S. Pradhan, "A transition-based algorithm for AMR parsing," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics*. Denver, Colorado : Association for Computational Linguistics, 2015, pp. 366–375.

[83] Y. Puzikov, D. Kawahara, and S. Kurohashi, "M2L at SemEval-2016 task 8 : AMR parsing with neural networks," in *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*. San Diego, California : Association for Computational Linguistics, Jun 2016, pp. 1154–1159. [Online]. Available : https://www.aclweb.org/anthology/S16-1178

[84] Y. Artzi, K. Lee, and L. Zettlemoyer, "Broad-coverage CCG semantic parsing with AMR," in *Proceedings of the 2015 Conference on Empirical*

Methods in Natural Language Processing. Lisbon, Portugal : Association for Computational Linguistics, September 2015, pp. 1699–1710. [Online]. Available : https://www.aclweb.org/anthology/D15-1198

[85] X. Peng, L. Song, and D. Gildea, "A synchronous hyperedge replacement grammar based approach for AMR parsing," in *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*. Beijing, China : Association for Computational Linguistics, Jul 2015, pp. 32–41. [Online]. Available : https://www.aclweb.org/anthology/K15-1004

[86] M. Pust, U. Hermjakob, K. Knight, D. Marcu, and J. May, "Parsing English into Abstract Meaning Representation using syntax-based machine translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal : Association for Computational Linguistics, Sep 2015, pp. 1143–1154. [Online]. Available : https://www.aclweb.org/anthology/D15-1136

[87] H. Cartan, "Théorie des filtres," in *C. R. Acad. Sci. Paris*. Académie des sciences, 1937, vol. 205, pp. 595–598.

[88] B. Meyer, "Applying "design by contract"," *IEEE Computer*, vol. 25, no. 10, pp. 40–51, October 1992.

[89] R. B. Findler, M. Latendresse, and M. Felleisen, "Behavioral contracts and behavioral subtyping," in *ACM Conference Foundations of Software Engineering*, 2001.

[90] C. A. R. Hoare, "An axiomatic basis for computer programming," *Communications of the ACM Volume 12 / Number 10 / October, 1969*, vol. 12, no. 10, pp. 576–583, 1969.

[91] P. W. O'Hearn, "Separation logic," *Commun. ACM*, vol. 62, no. 2, pp. 86–95, 2019. [Online]. Available : https://doi.org/10.1145/3211968

[92] N. A. Lynch and E. W. Stark, "A Proof of the Kahn Principle for Input/Output Automata," *Inf. Comput.*, vol. 82, no. 1, pp. 81–92, 1989.

[93] N. A. Lynch, "Input/output automata : Basic, timed, hybrid, probabilistic, dynamic," in *CONCUR*, 2003, pp. 187–188.

[94] A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and F. Y. C. Mang, "Synchronous and Bidirectional Component Interfaces," in *Proc. of the 14th International Conference on Computer Aided Verification (CAV'02)*, ser. Lecture Notes in Computer Science, vol. 2404. Springer, 2002, pp. 414–427.

[95] L. Lamport, "What good is temporal logic ?" *Information Processing, R. E. A. Mason, ed., Elsevier Publishers*, vol. 83, pp. 657–668, May 1983. [Online]. Available : https://www.microsoft.com/en-us/research/publication/good-temporal-logic/

[96] A. Arnold and D. Niwiński, *Rudiments of [mu]-calculus*, ser. Studies in Logic and the Found. Elsevier, 2001.

[97] A. Arnold, A. Vincent, and I. Walukiewicz, "Games for synthesis of controllers with partial observation," *Theoretical Computer Science*, vol. 303, no. 1, pp. 7–34, 2003, logic and Complexity in Computer Science.

[98] Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, Aug 1986.

[99] F. Moller and S. Smolka, "On the computational complexity of bisimulation," *ACM Comput. Surv.*, vol. 27, pp. 287–289, 06 1995.

[100] S. R. Boudaoud, "Spécification d'exigences en langue naturelle, avec automates et logique," Master's thesis, Université de Rennes 1, 2016, private communication.

[101] A. Lamercerie, "Analyse formelle d'exigences en langue naturelle pour la conception de systèmes cyber-physiques," in *Actes de la Conférence TALN*, vol. 2. ATALA, May 2018, pp. 41–54. [Online]. Available : https://www.aclweb.org/anthology/2018.jeptalnrecital-recital.4

[102] K. Gan-Krzywoszyńska, "La connexion syntaxique," *Philosophia Scientia*, vol. 11-2, pp. 97–120, 2007.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ DE RENNES 1

Titre : Principe de transduction sémantique pour l'application de théories d'interfaces sur des documents de spécification

Mot clés : Traitement automatique du langage naturel, Représentation sémantique, Ingénierie des exigences, Raisonnement par contrats, Algèbre de composition

Résumé : La spécification de systèmes techniques est une tâche complexe et source d'erreurs. D'un point de vue méthodologique, les caractéristiques attendues doivent être rigoureusement spécifiées. En pratique, des cahiers des charges regroupent les propriétés voulues sous la forme d'une liste de conditions à vérifier, nommées exigences. L'enjeu de cet ouvrage est la construction d'un processus d'analyse pour une application sur des documents textuels, rédigés dans une langue naturelle telle que l'anglais. La mise en œuvre visée est une chaine de traitement, automatisée de bout en bout, et intégrant des facultés d'interprétation et de raisonnement sur les données traitées. Précisément, nous proposons d'étudier comment relier des énoncés de langue anglaise à des modèles formels exploitables dans un cadre théorique adapté. Premièrement, le principe de transduction sémantique est avancé pour extraire et formaliser des énoncés du langage naturel. Dans un second temps, les propriétés algébriques de modèles de spécification sont étudiées pour définir une théorie permettant de vérifier la consistance des exigences d'un cahier des charges.

Title: Semantic Transducer for Interface Theories

Keywords: Natural Language Processing, Meaning Representation, Requirements Engineering, Contract Theory, Compositional Algebra

Abstract: The specification of technical systems is a complex and error-prone task. From a methodological point of view, the expected characteristics must be rigorously specified. In practice, specifications group together the desired properties in the form of a list of conditions to be checked, called requirements. The goal is to build an analysis process for application on textual documents, written in a natural language such as English. The targeted implementation is an end-to-end automated processing chain, integrating faculties of interpretation and reasoning on specification data. Precisely, we propose to study and experiment how to link natural language statements to formal models that can be exploited in a theoretical framework. First, the principle of semantic transduction is advanced to extract and formalize natural language statements. In a second step, the algebraic properties of specification models are studied to define a theory to verify the consistency of requirements.